

Python 自动化测试实战

无涯 著

電子工業出版社

Publishing House of Electronics Industry

北京 · BEIJING

内 容 简 介

本书结合大量实际应用的案例，重点讲解了自动化测试在企业级的应用技术和实战。本书帮助读者系统地学习 UI 自动化测试和接口自动化测试的技能。

本书分为两大部分：第 1 部分 UI 自动化测试，包括第 1~8 章，以新的视角介绍了 Selenium 的知识体系，结合具体的案例讲解了 Selenium 的 API 在 UI 自动化测试的不同场景下的应用，典型问题的解决思路，主要内容包括 Selenium 的 API、单元测试框架 unittest、Jenkins、数据驱动、页面对象设计模式和 UI 自动化测试实战。第 2 部分接口自动化测试，包括第 9~13 章，介绍了 HTTP 应用层的协议，序列化与反序列化的知识，以及主流的测试工具 PostMan、JMeter 和 Requests 库在接口自动化测试中的应用和案例。

相信本书能够帮助想学习自动化测试的读者，以及准备带领团队进行自动化测试转型的测试管理者学习和掌握自动化测试实战技能。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

Python 自动化测试实战 / 无涯著. —北京：电子工业出版社，2019.3

ISBN 978-7-121-35704-6

I. ①P… II. ①无… III. ①软件工具—程序设计IV. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 271146 号

策划编辑：张瑞喜

责任编辑：张瑞喜

印 刷：中国电影出版社印刷厂

装 订：中国电影出版社印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：710×1000 1/16 印张：20.5 字数：367 千字

版 次：2019 年 3 月第 1 版

印 次：2019 年 3 月第 1 次印刷

定 价：69.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：zhangruixi@phei.com.cn。

序 1

技术赋能未来，推进个人成长

从人类第一次工业革命开始，技术就成为推进革新的原始驱动力之一，蒸汽机的出现促使了手工业到机器工业的转变，这是一场以机器替代人工的革命。在信息技术发展飞速的今天，技术更是变成了提高工程生产率的加速剂。测试是一个技术性的行业，投资自己的技术，赋能自己的未来。

当今行业内推崇测试的技术和技术的测试，其中，测试的技术就是各种可以提高测试可靠性、可持续性、可重复性和可复现性的技术手段，这其中包含了 IT 行业都在做的自动化测试、持续集成、持续交付、DevOps；而技术的测试是指在上述过程中通过技术手段完成具体工作内容的技术方法或者落地技术手段，既可以是测试辅助工具，也可以是为了测试而撰写的测试脚本。但是无论是哪一种，都是为了能够将测试工程师的时间释放出来，让机器完成重复可循环的工作内容，将时间还给测试工程师去完成创造性的工作内容。在任何一个团队，降低成本、提高效率都是一个永恒的话题，在测试行业里，自动化测试技术无疑是一个提高效率很好的工具。通过自动化将模式化、重复化的人类劳动变成机器的逻辑复现，释放人工，提高生产率。

近些年来，自动化测试、测试脚本的编写、测试框架的使用或者打造几乎变成公司招聘测试工程师的标配职位要求。但是，在实际操作过程中，测试工程师大量时间又被人工的重复工作内容所占据。这就变成了一对矛盾体。矛盾是由两个方面决定的，矛盾的一个方面如果不存在，另外一个就自然而然也不存在了。测试工作者通过学习和实践，将工作中繁重的手工劳动变成自动化的机器循环，就从根本上解决了上述矛盾，提升了效率，提升了自我。

随着自动化测试技术逐渐变成测试工程师的基础性要求，越来越多的 IT



技术从业者都开始着手学习自动化的测试技术，无论是 UI 的自动化，还是 API 的自动化，所有人都希望通过自动化的方式来完成枯燥乏味的、无限循环的手工测试劳动，这时需要先选择一门语言。尽管 Java 语言在研发领域占领主要地位，但是对于测试来说，并不一定要选择 Java。这是因为 Java 入门门槛较高，入门和掌握这门语言需要花费更多的时间。

现在中小学教育已经加入了 Python 的课程，这也预示了 Python 的学习门槛会比较低，入门简单、容易掌握。那么通过 Python 入手开始学习编程技术，对于绝大部分行业新人来说是一个明智的开始。

技术赋能未来，推进个人成长。任何时候开始学习都为时不晚。如果您对任何一门编程语言都不是很熟悉，那么选择 Python 会让您更快速地走上自动化测试的道路。对于测试工程师，选择 Python 会大大缩短您从第一行代码到自动化脚本的学习时间。

本书由浅入深地介绍基于 Python 的自动化测试，从基础框架讲解、基础 API 使用，到框架设计、模式引入、持续集成等。通过本书的学习，您既可以掌握基于 Python 语言的自动化测试的技术解决方案，以及相关的工程化解决思路，同时还可以快速搭建出属于您自己的测试平台。

作者本着在实践中学习，在学习实践中实践的思维模式，设计了每一个章节的例子，并提供了案例的相关源代码，这对于初学者的学习是非常有帮助的。

对于不想被技术创新的时代车轮丢下的测试工程师和测试管理者，这本书值得一读。

陈磊（测者）

京东商城测试架构师

2018 年 11 月于北京

序 2

我已经连续两年组织了 QA Intelligence 发布《软件测试行业现状报告》的中文翻译工作，在 2018 年的报告中，有两段话让我印象深刻：

“越来越多的人从公司的 IT 和非 IT 部门转行测试，这表明测试似乎已经成为一个令人感兴趣的职业发展方向。”

“测试人员做的不仅仅是测试应用程序，76%的人还会进行自动化测试和脚本编写。”

相信这与大部分业内人士对当前软件测试行业的看法是一致的：

越来越多的人进入软件测试行业，越来越多的公司开展自动化测试。

同时，我们也看到软件测试行业面临如下问题：

- 越来越多的人进入软件测试行业，但是，软件测试人员的 IT 技能却普遍不高；
- 越来越多的公司开展自动化测试，但是，自动化测试的成效却难以显现。

一个很有意思的现象是，几乎每一个测试人员在应聘的时候都会被要求具备自动化测试的相关技能，笔试或者面试的时候也会被问及自动化测试的相关问题，但是，入职后，却发现日常的工作依然是“点点点”，说好的自动化测试在巨大的交付压力和频繁的业务变化面前，常常因难以实施，不见成效，而无人提及。

有一些公司就更有意思了，组织大家写了很多自动化测试用例，最后，由于维护成本高昂，默默地丢弃了；还有的公司，写了一堆自动化测试案例，每月甚至每几个月才运行一次，每次都要几日甚至数周的时间才能磕磕绊绊地运行完毕，并且需要由专门的开发测试人员照看着这些“资产”，稍有应用修改就需要大量调整案例和脚本，投入与产出严重不成比例。

以上，我称之为“软件测试中之怪现象”。

究其原因，在于从业者的技能不高，在于管理者的推进方法不当。



说到自动化测试，对于测试人员来说，接触最多的就是功能自动化测试和接口自动化测试了。从近几年与不少测试从业者的交流中可以看出，这的确也是当前很多测试人员提升自身技能的一个不错的切入点。

本书作者无涯从事软件测试工作多年，在功能和接口自动化测试方面多有心得，并且乐于分享，笔耕不辍，其公众号“Python 自动化测试”帮助了很多软件测试学习者，很多学习者反馈其公众号中的文章对自身技能提升助益良多。

与无涯的相识，还是源于一年前张瑞喜老师的引荐。日后多有交流，日渐熟稔。受邀为其新书作序，有幸得其新作书稿先睹为快。作者的文字平实，重干货，少浮华摘录，内容更偏重在功能和接口自动化测试的实战讲解。全书从 Selenium 到 unittest，从 PostMan 到 JMeter，中间还穿插持续集成和数据驱动等相关话题，内容翔实，图文并茂，相信本书能够帮助多年一直在做测试并想转型做自动化测试的读者，以及准备带领团队进行自动化测试转型的测试管理者学习和掌握自动化测试实战技能。

相信此书对于希望提升自身自动化技能水平的软件测试学习者一定会大有裨益。

阿 奎

《Python 编程基础与 HTTP 接口测试》作者

2018 年 11 月 7 日

前言

“学海无涯苦作舟”是我特别喜欢的一句话，所以我的网名叫“无涯”。因为自认为没有出众的天赋和才华，所以需要不断地鞭策自己。学习是一个痛苦的过程，某些时候感觉就像西西弗斯一样日复一日地干着同样一件事，明知很累但是还要不断地重复。

写书是一件很偶然的事。开始在百度阅读平台写文章，后来在公众号上不断地更新文章，把自己这几年学到的知识记录下来并分享给网友。出这本书，并不是说自己有多么专业，恰恰相反，我始终感觉像在无边的大海中一样，自己掌握的知识可以忽略不计。但我还是希望这些点点滴滴的记录能够帮助业界的同仁，帮助读者更深入地理解功能自动化测试和接口自动化测试的知识体系，掌握自动化测试的实战技能。

一直以来，很多测试人员想学习自动化测试技术但是又感到比较迷茫。一方面许多公司要求测试人员掌握自动化测试的技术；另外一方面，一些人虽然已经具备自动化测试技术，但是却感觉自动化测试在实际工作中的应用和推广比较艰难，很难取得测试效率的提升并得到管理层的认可。

工作中一路走来，我深知自动化测试技术对测试人员的重要性，以及在产品测试中的重要性。公司对产品的迭代速度越来越快，而且要求以更快的速度、更高的质量把产品推向市场，给客户带来价值。在这个层面上，测试是不能马虎的，产品上线后，一旦出现产品质量上的问题，则不仅没有给用户带来应有的价值，还会影响产品的口碑。

技术应该为人服务，同理，测试技术应该为测试人员服务。所以我们应该应用开源的测试工具和主流的开发语言，结合相关的测试技术来提高测试效率，

从而提高工程效率，这是测试价值的一个很重要的体现方面。目前，测试的核心是手工测试，但自动化测试是测试人员的未来。技术是无法取代人的，但技术可以代替大量的人工。自动化测试技术可以很好地协助我们做回归测试，以及应用接口自动化测试的技术来提高产品质量测试的效率。测试人员应该更加关注应用层程序的交互和产品之间的业务流程的调用，而不是花费大量的精力在浅显的功能测试中。另外一个方面，当测试技术的维护成本高于我们自己能力的控制范围时，则测试人员和技术管理者需要思考针对测试技术的策略是否存在问题。自动化测试技术是一种能力，同时也是提高工程效率很重要的一部分。

我一直认同零缺陷的思想，它倡导我们做有价值的事。要体现出自动化测试技术的价值，就要使之确确实实地帮助团队提高产品测试效率，解决测试团队中人力投入过大的问题，让测试团队把更多的心思来放在测试设计和测试策略上来。

本书面向的读者

本书特别适合如下读者：

- (1) 想转自动化测试的测试工作者；
- (2) 已有一定 Python 基础知识想学习自动化测试者；
- (3) 已有一定自动化测试的基础知识并想系统地了解自动化测试的读者；
- (4) 想系统了解接口自动化测试在企业级的实战案例的读者；
- (5) 测试团队管理者。

如何阅读本书

首先很感谢您阅读我写的第一本书。在本书中，系统完整地介绍了 Python 语言在功能自动化测试和接口自动化测试中的应用。本书要求读者具备一定的 Python 语言基础和测试方面的基本知识背景。书中对很多代码都增加了注释，方便理解。在功能自动化测试部分中，详细地介绍了 Selenium 的 API 在 Web 自动化测试中各个场景下的应用，也增加了一些产品的实战来说明 Selenium 的框架在 Web 自动化测试中的实战应用。在接口自动化测试部分，主要知识点包含了 HTTP 协议，主流测试工具 PostMan 和 JMeter 在接口自动化测试中的实例

应用和实战案例，最后书中还介绍了 Python 语言结合第三方的库（Requests）在接口自动化测试中的应用，以及接口测试框架的设计、接口测试在企业级的实战。

建议读者跟着书中的案例一步一步地进行练习，举一反三地把学习到的知识应用到具体的测试实践中，并在不断的总结中把书本知识变为属于自己的知识。

感谢

首先感谢张瑞喜老师在这一年多中对我的支持和鼓励，才让我有勇气来完成这本书的写作。

借这个机会，特别感谢曾给予我帮助的领导和同事！

当然，在此还要感谢“Python 自动化测试”公众号的粉丝和学员，感谢他们的信任和支持。谢谢参与审校的同学，他们分别是雷夏阳、郑芯婷、郭志峰、王天亮、赵锦涛、段惠艳、王燕。

最后感谢我的家人对我的陪伴，谢谢他们对我的包容和理解。

无涯

2018 年 11 月 西安



目 录

第 1 部分 UI 自动化测试

第 1 章	自动化测试概述	2
1.1	自动化测试的价值	2
1.2	自动化测试的应用	3
第 2 章	Selenium 元素定位实战	5
2.1	Selenium 简述	5
2.2	Selenium 结合浏览器实战	5
2.3	元素定位实战	6
第 3 章	Selenium 与页面的交互	24
3.1	WebDriver 浏览器的属性	24
3.2	WebElement 类的方法	29
3.3	下拉框实战	35
3.4	弹出框实战	39
3.5	WebDriverWait 类实战	44
3.6	ActionChains 类实战	50
3.7	键盘事件实战	54
3.8	JavaScript 的处理	55
3.9	获取截图	60



第 4 章	单元测试框架 unittest	62
4.1	unittest 简述	62
4.2	测试固件	63
4.3	测试执行	67
4.4	构建测试套件	70
4.5	分离测试固件	76
4.6	测试断言	78
4.7	断言的注意事项	82
4.8	批量执行测试用例	84
4.9	生成测试报告	87
4.10	代码覆盖率统计实战	89
第 5 章	Jenkins 实战	92
5.1	Jenkins 简述及部署	92
5.2	Jenkins 实战	92
第 6 章	数据驱动	105
6.1	ddt 实战	105
6.2	Txt 实战	108
6.3	Csv 实战	111
6.4	Excel 实战	114
6.5	Xml 实战	119
6.6	MySQL 实战	123
第 7 章	Page Objects 实战	131
7.1	Page Objects 的实现	131
7.2	Page Objects 中引入 wait	140
7.3	Page Objects 引入工厂设计模式	142

第 8 章	UI 自动化测试实战	149
8.1	Web 产品的实战	149

第 2 部分 接口自动化测试

第 9 章	HTTP 协议	162
9.1	HTTP 简述	162
9.2	HTTP 的状态码	163
9.3	Cookie 的请求流程	175
9.4	Session 的请求流程	178
9.5	Token 的请求流程	180
第 10 章	序列化与反序列化	183
10.1	JSON 库的应用	183
10.2	JSON 库的实例实战	185
第 11 章	PostMan 的应用	190
11.1	PostMan 简述	190
11.2	PostMan 实战	191
11.3	PostMan 接口测试实战	219
第 12 章	JMeter 接口测试应用	226
12.1	JMeter 简述	226
12.2	JMeter 的语言切换	226
12.3	JMeter 的插件安装	226
12.4	WebServices 的请求	227
12.5	HTTP 的请求	229

12.6	JMeter 断言实战	232
12.7	HTTP 请求默认值	234
12.8	用户定义的变量	235
12.9	Token 的获取实战	237
12.10	HTTP Cookie 管理器实战	240
12.11	生成测试报告实战	244
12.12	自动发送邮件实战	248
12.13	引入 CI	252
12.14	JMeter 接口测试实战	256
第 13 章	Requests 实战	263
13.1	Requests 简述	263
13.2	Requests 发送请求	264
13.3	URL 参数实战	266
13.4	请求头的添加	267
13.5	data 参数实战	268
13.6	JSON 参数实战	270
13.7	Token 实战	272
13.8	Session 实战	276
13.9	Session 会话对象	281
13.10	Requests 鉴权实战	286
13.11	超时处理	287
13.12	文件下载	289
13.13	文件上传	293
13.14	Requests 接口测试实战	296
	主要参考文献	314



第1部分 UI自动化测试

第1章 自动化测试概述

第2章 Selenium元素定位实战

第3章 Selenium与页面的交互

第4章 单元测试框架unittest

第5章 Jenkins实战

第6章 数据驱动

第7章 Page Objects实战

第8章 UI自动化测试实战

第 1 章 自动化测试概述

1.1 自动化测试的价值

传统的商业模式以业务驱动产品，而现在则以技术驱动产品。特别是在产品迭代速度快，市场不断变化的当下，产品调整，很多时候是基于客户的需求，基于整个产品战略的调整。单纯的手工测试越来越无法适应这个变化的过程。测试人员怎样做到快速响应并且保证产品上线后质量能够满足市场的要求？怎样通过测试技术的手段来达到测试效率的提升？这些值得我们思考。

测试工作的意义，主要包括两个方面，一是产品质量的管理，二是测试效率的提升。

自动化测试技术的应用越来越普遍，这源于企业的要求和互联网技术的发展。测试人员，已经不能按过去流水线式的方式进行测试，因为这种测试效率低下的生产方式会逐渐被现代企业所淘汰。企业要在新一轮的竞争中脱颖而出，必须快速推出新的产品，并以更快的方式占领市场。测试人员需要考虑的是，怎样快速适应这种节奏的变化，更快地完成产品的测试，并且使产品质量满足发布要求。在这个过程中，可以说自动化测试技术是测试人员的必然选择，也是技术发展的选择。

站在企业管理层的角度来说，自动化测试技术能够提升团队的战斗力，降低人力成本。对于持续迭代的产品，随着产品的增加，测试人手开始显得紧张，而增加人手则意味着人力成本大幅上升。那么，通过什么样的方式来解决这个问题呢？自动化测试技术的引入可以很好地解决这个问题。在一个可持续迭代的产品测试体系中，完全可以使用自动化测试技术来实现系统已有的功能测试，即使无法达到百分之百实现，实现百分之五十也是很好的，例如，测试环境合并代码后

的回归测试，上线后的冒烟测试^①等。而测试人员只需要测试新的功能点和自动化测试未实现的功能点，这样就大大地提高了整个测试的效率。

1.2 自动化测试的应用

测试人员需要思考如何提升测试效率，如何让自动化测试技术应用在产品测试中，从而给自己留出更多的时间来思考产品质量策略和新的测试场景。在笔者看来，不管是个人还是团队，既然考虑引进自动化测试的技术，就要把它应用起来。笔者也曾经见过有些团队把产品自动化测试做起来后，由于维护成本大又放弃了。其实，任何技术，都既存在优点也存在缺陷，没有一个测试框架能够解决所有的问题，我们应该把它应用到合适的场景中，让它带给我们想要的效果。

1. UI 自动化测试的应用

对于产品迭代速度很快的公司，特别是互联网公司，使用 UI 自动化测试技术并不是一个很理想的选择。这主要是由于互联网产品迭代速度太快，使用 UI 自动化就会陷入不断的维护之中。在产品上线前，用 UI 自动化测试用例进行测试的执行时间很长，导致人员必须等待。所以针对这种类型的产品测试，可以使用 UI 自动化测试只覆盖主要流程的方法，只对产品的核心流程进行 UI 自动化测试，而不覆盖大多数测试点。

针对迭代速度不是很快、开发周期比较长的产品，使用 UI 自动化测试是一种比较好的选择，它可以把能够覆盖到的场景都覆盖到。在实际的测试过程中，如果在晚上下班的时候自动执行 UI 自动化测试用例，则第二天早上就可以看到自动化测试的执行结果，依据自动化测试报告信息，就能够得到本次迭代产品的质量情况。在下次迭代中依此循环，直到产品（项目）测试结束。

2. 接口自动化测试的应用

前面提到，当产品迭代速度很快时，使用 UI 自动化测试的技术并不是一个理想的选择。在测试的金字塔模型中，第一层是 UI 层，第二层是 API 层，第三

^① 冒烟测试：冒烟测试是指完成一个版本的产品开发后，对该版本产品最基本的功能进行测试，保证基本功能正常和流程能走通。也指产品上线之后，对最基本的业务流程进行的验证和测试。

层是 UNIT 层。当产品迭代速度快时，选择 API 层进行测试是一种比较好的方案。

另外，基于目前，的开发模式基本都是前后端分离的，做接口测试也是一种比较好的选择。选择接口测试技术还有一个优势是，接口相对来说比较稳定，因此测试的效率比较高。测试时应该把更多的精力和时间放在客户端与服务端之间的交互上，放在前后端分离的模式中。平常我们所测试到的产品（WEB、APP、PAD 等类型）都是客户端产品，这样我们只需要更多地关注客户端与服务端的业务交互和逻辑处理。测试接口时，不能仅仅测试客户端与服务端之间的通信，这样完全不能满足测试的需求。在接口测试时，有两个维度是必须要考虑的。第一是 API 的内部逻辑。这主要包含异常处理的测试。通过测试来验证服务端程序的稳定性、健壮性及容错性，测试点的选择需要考虑的主要是请求参数的字段类型、字段参数中不能为空的字段验证，以及请求参数边界值的验证。第二是接口功能的完整性。通俗地说，是通过接口测试技术来测试产品的业务逻辑和流程。也就是说，首先要保证产品的业务流程是好的，然后才考虑产品的性能测试和安全测试。可以使用主流的测试工具（PostMan, JMeter），或者 Python 语言代码来进行产品的接口自动化测试。这样，不管是在测试阶段还是在上线阶段后的冒烟测试，接口测试的执行速度是非常快的，即使几百个的接口测试用例，也只需要几分钟就会执行完毕。

微服务目前已不是一件很新鲜的事了，很多公司目前都在应用。我认为它逐步会发展成测试的主要趋势。在微服务中，以一组 API 提供业务功能的组件；微服务之间是以 HTTP 等轻量级的通信方式进行调用。那么针对微服务的测试，需要做的就是了解微服务的内部请求方式，然后通过应用层的协议来测试它。当然这中间还涉及契约测试。测试需要保障每个微服务组件的内部功能是正确的，同时还要保障每个微服务组件之间连接的正确性。微服务测试还需要注意测试微服务提供的功能是否能够满足他人的需求，例如，开放平台的微服务能否满足第三方的调用和业务需求。

第 2 章 Selenium 元素定位实战

2.1 Selenium 简述

Selenium 是一个用于 Web 应用程序的自动化测试工具。Selenium 直接运行在浏览器中，它可以模拟用户的行为操作，操作界面友好。Selenium 支持 IE、Google Chrome、Firefox、Opera 等主流浏览器，同时 Selenium 也支持主流开发语言，如 Java、Python、C#等。本书主要介绍基于 Python 语言，结合 Selenium 进行产品自动化测试的内容。目前，一些主流浏览器厂商已经采取措施使 Selenium 成为浏览器的一部分，厂商还提供了不同的驱动程序（Driver）来兼容 Selenium 的版本。目的是使浏览器在执行程序时更加稳定。

2.2 Selenium 结合浏览器实战

针对不同的浏览器，需要下载不同的驱动程序（Driver）。以 Firefox 浏览器为例，在 <https://github.com/mozilla/geckodriver/releases/> 网站中选择 “geckodriver - v0.23.0-win64.zip” 下载并解压文件后，把 Geckodriver.exe 文件放在 Python 安装程序的目录下，也就是 C:\Python36-32。切记 Firefox 浏览器的版本号一定要大于 48，否则运行代码后会出现驱动程序（Driver）版本与 Firefox 浏览器不兼容的情况。针对 Firefox 浏览器的测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver


driver=webdriver.Firefox()
```

```
driver.get('http://www.baidu.com/')
driver.find_element_by_id('lst-ib').send_keys('Selenium')
driver.quit()
```

2.3 元素定位实战

在 UI 自动化测试中，最基础最核心的技能是对页面元素进行定位，定位到相应的元素后才可以对页面的操作进行编码验证。

2.3.1 调试工具实战

在 Chrome 浏览器中，点击鼠标右键，在弹出的快捷菜单中选择“检查”选项；在弹出的调试信息窗口中，点击  按钮后^①，将鼠标移动到需要定位的目标位置，调试信息窗口中就会显示元素的属性。


以对百度搜索页面的测试为例，在调试信息窗口中点击  按钮后，将鼠标移动到百度搜索输入框上，屏幕上就会显示元素属性。如图 2-3-1 所示。



图 2-3-1

在图 2-3-1 中可以看到，百度搜索输入框的元素属性 ID 为 kw，NAME 为 wd，CLASS_NAME 为 s_ippt。

① 注：在不同的浏览器中，这一按钮的形状可能稍有差别

在浏览器中点击菜单栏的“工具”选项卡，在弹出的“工具”选项中点击“Web 开发者”选项中的“Web 控制台”子选项，如图 2-3-2 所示。

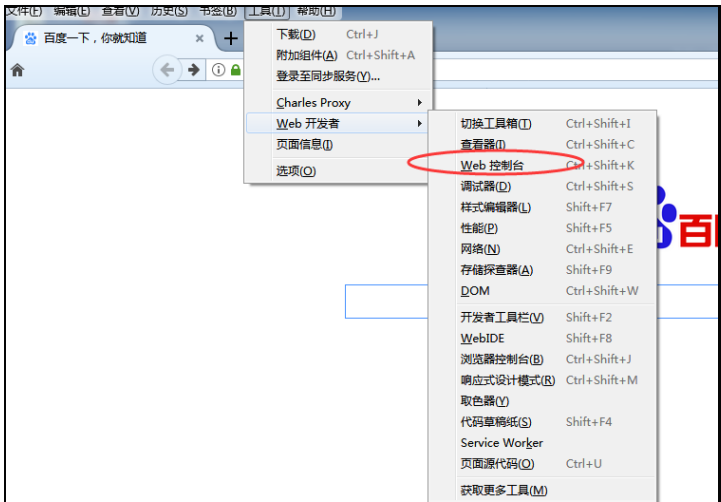


图 2-3-2


这里依然以对百度搜索页面的测试为例。点击 Web 控制台的  按钮后，将鼠标移动到搜索输入框上，屏幕上显示百度搜索输入框的元素属性，如图 2-3-3 所示。



图 2-3-3

2.3.2 单个元素定位实战

在 Selenium 自动化测试中，提供了单个元素定位方式和多个元素定位方式。两种方式都是根据元素属性 ID、NAME、CLASS_NAME、TAG_NAME、CSS_SELECTOR、XPATH、LINK_TEXT、PARTIAL_LINK_TEXT 来进行定位。下面就通过具体的实例说明单个元素定位在 UI 自动化测试中的应用。

1. find_element_by_id

通过元素属性 ID 定位到元素，方法是 find_element_by_id。这里以百度搜索输入框为例，它的代码是：

```
<input id="kw" name="wd" class="s_ipt" value="" maxlength="255"
autocomplete="off">
```

它的 ID 属性是 kw，在百度搜索输入框中输入搜索的关键字“selenium”的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_id('kw').send_keys('selenium')
driver.quit()
```

2. find_element_by_name

通过元素属性 NAME 定位到元素，方法是 find_element_by_name。它的 NAME 元素属性是 wd，在百度搜索输入框中输入中实现搜索关键字的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
```

```
driver.get('http://www.baidu.com')
driver.find_element_by_name('wd').send_keys('selenium')
driver.quit()
```

3. find_element_by_class_name

通过元素属性 CLASS_NAME 定位到元素，方法是 find_element_by_class_name。还是以在百度搜索输入框中输入搜索关键字“selenium”为例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_class_name('s_input').send_keys('selenium')
driver.quit()
```

4. find_element_by_xpath

通过 XPATH 定位百度搜索输入框的元素，方法是 find_element_by_xpath，原始属性是//*[@id="kw"]。获取的方式是定位到百度搜索输入框的元素属性后，用鼠标右键点击该属性，在弹出的快捷菜单中上选择“Copy”选项，在“Copy”子选项中选择“Copy Xpath”选项，如图 2-3-4 所示。

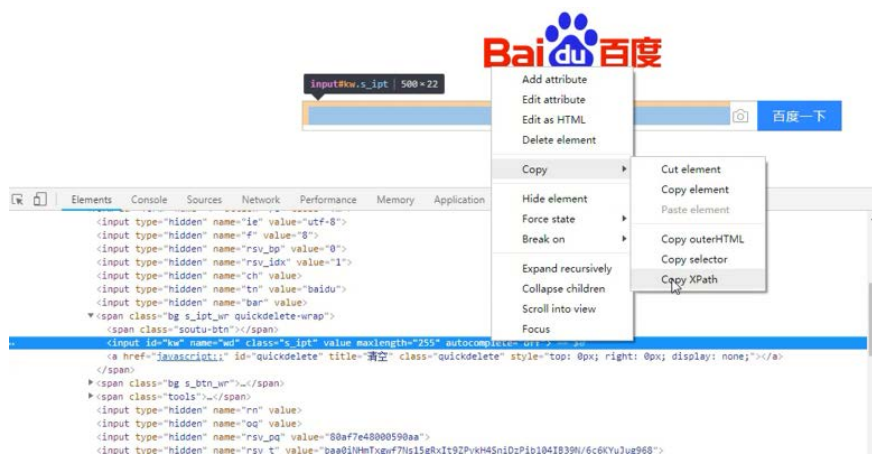


图 2-3-4

以在百度搜索输入框中输入搜索关键字“selenium”为例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_xpath('//*[@id="kw"]').send_keys('selenium')
driver.quit()
```

5. find_element_by_link_text

LINK_TEXT 用于对超链接的处理。在 HTML 的代码中主要是以标签 a 对应，方法是 find_element_by_link_text。以点击百度首页的“新闻”链接为例，查看“新闻”对应的代码：新闻。依据代码可以看到它是以 a 标签的。下面实现点击百度首页的“新闻”链接，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_link_text(u'新闻').click()
driver.quit()
```

6. find_element_by_partial_link_text

PARTIAL_LINK_TEXT 也用于对超链接的处理，它与 LINK_TEXT 不同的是，它是按模糊搜索方式处理的。例如，在百度首页包含“闻”字的只有新闻链接，那么操作的时候只需要填写“闻”字就可以定位成功，方法是 find_element_by_partial_link_text。仍以实现点击百度首页的“新闻”链接为例，代码

如下:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_partial_link_text(u'闻').click()
driver.quit()
```

7. find_element_by_css_selector

当使用 ID、NAME 等方式定位不到元素的时候,可使用 CSS_SELECTOR,方法是 find_element_by_css_selector。这里以百度搜索输入框为例,获取的方式是定位到百度搜索输入框的元素属性后,用鼠标右键点击该属性,在弹出的快捷菜单中上选择“Copy”选项,在“Copy”子选项中选择“Copy selector”选项,就可以获取到基于 CSS_SELECTOR 的元素属性是#kw,如图 2-3-5 所示。

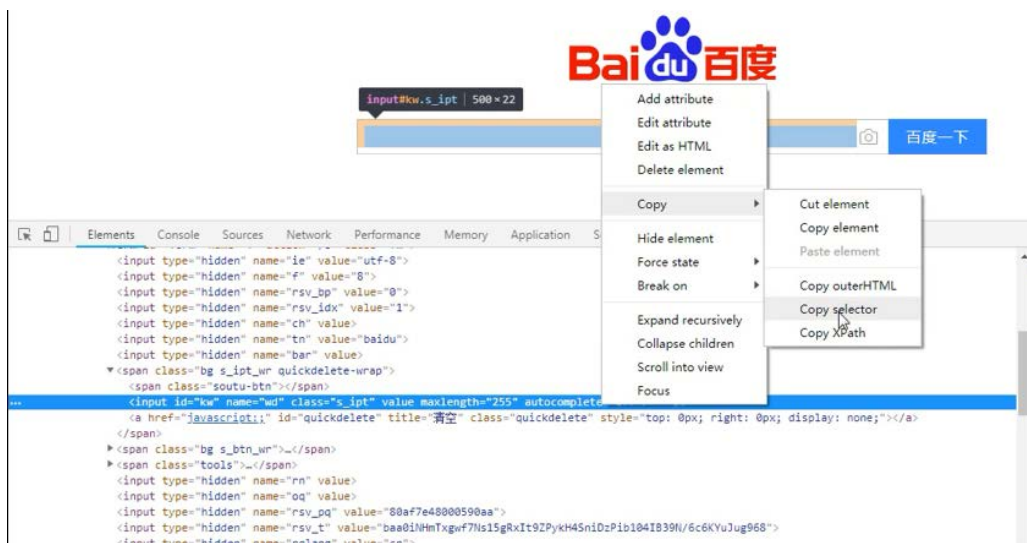


图 2-3-5

在百度搜索输入框中输入搜索关键字“selenium”的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_element_by_css_selector('#kw').send_keys('Selenium')
driver.quit()
```

2.3.3 多个元素定位实战

在工作中，某些时候可能会发现元素的 ID、NAME、CLASS_NAME 等元素属性是一致的，这时，使用 ID、NAME、CLASS_NAME 等这些元素属性定位时就无法准确地定位到具体的元素。例如，在百度首页中，以 TAG_NAME 元素属性来定位百度搜索输入框，在代码中发现不仅仅在百度搜索输入框有 input 标签，在百度搜索输入框之前也有 input 标签，这时该如何定位呢？这时可以使用多个元素定位的方式。当定位到多个元素后，结果将以列表形式呈现，然后可以按照列表的索引来定位到具体的元素位置。下面以 ID、TAG_NAME 为例进行讲解。

1. find_elements_by_tag_name

以百度搜索输入框为例，使用 TAG_NAME 的方式来实现定位，它的 TAG_NAME 是 input，首先来获取百度首页的 input 标签，并且查看它的类型，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
```

```
driver.get('http://www.baidu.com')
tag_names=driver.find_elements_by_tag_name('input')
for tag_name in tag_names:
    print tag_name
print type(tag_names)
driver.quit()
```

运行以上代码后输出的结果为:

```
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="fb4b2fa1-fa15-
408a-9c4d-4b6572804814")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="d92c3809-47de-
4472-b304-e87c04c247c7")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="2f2258f6-ad9d-
482b-952c-d98401eb5251")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="6824d195-3958-
48f3-942b-420451db1fe7")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="6977ddef-82f0-
4c90-8329-eb890c6bb5b0")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="906703ad-6e42-
4ad0-aaed-719f186df244")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="9c421db8-7e48-
44f1-9ee0-2efc6b7f8bee")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="6dc549fa-a1e7-
4811-8365-dc7561f8cd8e")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="5c5dd91b-b4d4-
4a70-bf98-aaa01ale55dd")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="d0550033-09e4-
4073-878a-3e244424324c")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="c05b5f40-3277-
492c-8e13-db5d711e5e29")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="29c8e081-eda1-
```

```

4f10-b395-1f2e4a37aa56")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="2d0ffa2c-8351-
46b1-aa4b-17a577b33326")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="5e9983da-2f3a-
4160-a23e-8e7528927320")>
<selenium.webdriver.firefox.webelement.FirefoxWebElement
(session="54cf7b2e-376d-489a-94d0-264fda35b081", element="a6189485-abf4-
4b77-b4e3-3818cd68a7db")>
<type 'list'>

```

注解：从以上代码和输出结果中，可以看到，input 的数据类型是 list，那么百度首页搜索输入框是在 input 标签中的第 8 位，也就是对应的索引是 7（索引是从 0 开始），如图 2-3-6 所示。

```

▼<form id="form" name="f" action="/s" class="fm">
  <input type="hidden" name="ie" value="utf-8">
  <input type="hidden" name="f" value="8">
  <input type="hidden" name="rsv_bp" value="0">
  <input type="hidden" name="rsv_idx" value="1">
  <input type="hidden" name="ch" value>
  <input type="hidden" name="tn" value="baidu">
  <input type="hidden" name="bar" value>
  ▼<span class="bg s_ipt_wr quickdelete-wrap">
    <span class="soutu-btn"></span>
    <input id="kw" name="wd" class="s_ipt" value maxlength="255"
autocomple="off"> == $0
    <a href="javascript:;" id="quickdelete" title="清空" class=
"quickdelete" style="top: 0px; right: 0px; display: none;">
    </a>
  </span>
  ▶<span class="bg s_btn_wr">...</span>

```

图 2-3-6

下面使用多个元素定位的方式，实现在百度搜索输入框中输入搜索关键字“selenium”。实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

```

```

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
driver.find_elements_by_tag_name('input')[7].send_keys('Selenium')
driver.quit()

```

2. find_elements_by_id

多个元素的定位思路都是一样的。下面通过一个具体的登录实例来看在 ID 一致的情况下，使用多个元素定位的解决方案。HTML 的代码如下：

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="success.html" method="post">
    <center>
        <p>用户名:<input type="text" id="login" name="username"></p>
        <p>密 码: <input type="text" id="login" name="passsword"></p>
        <p><input type="submit" id="login" value="登录"></p>
    </center>
</form>
</body>
</html>

```

在以上代码中可以看到用户名输入框、密码输入框，以及“登录”按钮的 ID 一致，现在希望通过脚本输入用户名和密码，点击“登录”按钮，跳转到 success.html 的页面，并且使用 ID 来定位。在以上代码中看到用户名和密码输入框，以及登录按钮的 ID 一致，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()

```

```

driver.implicitly_wait(30)
#index.html 的地址依据自己本地实际地址
driver.get('file:///D:/git/github/book/testCase/index.html')
ids=driver.find_elements_by_id('login')
#输入用户名
ids[0].send_keys('wuya')
#输入密码
ids[1].send_keys('admin')
#点击“登录”按钮
ids[2].click()
driver.quit()

```

2.3.4 By 类的分析

如果在定位元素属性中包含了如 ID 等元素属性，那么在一个测试中，元素定位具体有哪几种方式，可以在 by 模块中的 By 类中看到。By 类的代码如下：

```

class By(object):
    """
    Set of supported locator strategies.
    """

    ID = "id"
    XPATH = "xpath"
    LINK_TEXT = "link text"
    PARTIAL_LINK_TEXT = "partial link text"
    NAME = "name"
    TAG_NAME = "tag name"
    CLASS_NAME = "class name"
    CSS_SELECTOR = "css selector"

```

注解：在 By 类中，类属性实际就是元素定位的方式，经常使用的有 ID、NAME 等。

2.3.5 iframe 元素定位实战

在自动化测试中，如果无法定位到一个元素，那么最大的可能是定位的元素属性在 iframe 框架中。iframe 对象代表一个 HTML 的内联框架，在 HTML 中 iframe 每出现一次，一个 iframe 对象就会被创建。

1. 处理未嵌套的 iframe

iframe 存在嵌套和未嵌套的页面。首先来看未嵌套的页面，如图 2-3-7 所示。



图 2-3-7

实现以上截图效果的 HTML 代码如下。

其中 frame.html 代码为：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <center>
    <a href="frame-1.html" target="10">无涯课堂</a>
    <a href="frame-2.html" target="10">Python 接口测试实战</a><br><br>
  </center>
  <iframe id="text" src="frame-1.html" name="10-20" width="500"
height="170" align="center">

  </iframe>
</body>
</html>
```

frame-1.html 代码为：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
```

```
<body>
  <center>
    <font size=5>无涯课堂</font><br><br>
  </center>
  无涯课堂在网易平台已上线，欢迎大家关注<br>

</body>
</html>
```

frame-2.html 代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <center>
    <font size=5>Python 接口测试实战</font><br><br>
  </center>
  基于 Python 语言的接口自动化测试实战课程。
</body>
</html>
```

在图 2-3-7 中，想要获取 iframe 框架中的“无涯课堂”，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
from selenium.webdriver.common.by import By

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
#index.html 的地址依据自己本地实际地址
driver.get('file:///D:/git/github/book/html/frame/frame.html')
#获取"无涯课堂"文本信息
print driver.find_element_by_xpath('/html/body/center/font').text
driver.quit()
```

运行以上代码后，会出现如下的错误：

selenium.common.exceptions.NoSuchElementException: Message: Unable to locate

element: /html/body/center/font

依据给出的错误信息可以发现这个错误是元素定位错误导致的。由于存在 iframe 框架，首先需要进入到 iframe 框架，再定位 iframe 框架中的元素。定位的方式分为两种，一种是以 ID 的方式，另外一种是索引的方式。

以 ID 的方式定位时的实现代码：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
#index.html 的地址依据自己本地实际地址
driver.get('file:///D:/git/github/book/html/frame/frame.html')
#依据 iframe 的 Id 进入到 frame 框架
driver.switch_to_frame('text')
#获取"无涯课堂"文本信息
print driver.find_element_by_xpath('/html/body/center/font').text
driver.quit()
```

也可以通过索引的方式进入到 iframe 框架中，然后再定位对应页面的元素属性。在以上的实例中，只有一个 iframe，那么它的索引就是 0，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
#index.html 的地址依据自己本地实际地址
driver.get('file:///D:/git/github/book/html/frame/frame.html')
#依据索引进入到 frame 框架
driver.switch_to_frame(0)
#获取"无涯课堂"文本信息
print driver.find_element_by_xpath('/html/body/center/font').text
driver.quit()
```

2. 处理嵌套的 iframe

下面来看嵌套的 `iframe` 在自动化测试中如何进行元素定位，以及如何跳出嵌套。嵌套页面的效果如图 2-3-8 所示。

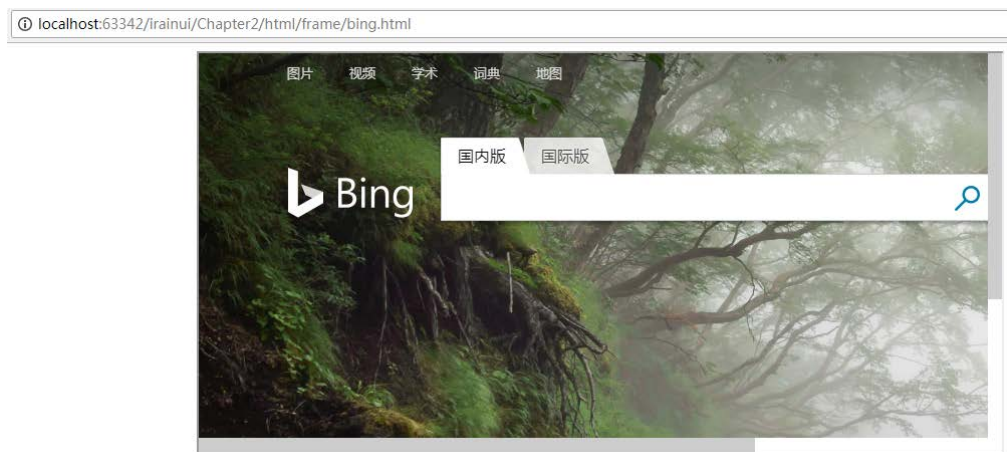


图 2-3-8

在图 2-3-8 中可以看到在一个页面里面嵌套了 Bing 搜索的首页，该页面的 HTML 代码如下：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>嵌套的页面</title>
</head>
<body>
<center>
    <iframe id="son" src="http://www.bing.com" width="800" height="400">

    </iframe>
</center>
<br><br>
</body>
</html>
```

在该页面要实现 Bing 的搜索效果，首先需要进入到 `iframe` 框架中，然后再定位 Bing 首页搜索输入框的元素属性。该 `iframe` 的 ID 是 `son`，Bing 首页搜索输入框的元素属性 ID 是 `sb_form_q`。实现在 Bing 首页搜索输入框中输入搜索关键

字的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
#bing.html 的地址依据自己本地实际地址
driver.get('file:///C:/Users/Administrator/Desktop/san/san/1130/frame/bi
ng.html')
#依据索引进入到 frame 框架
driver.switch_to_frame('son')
#输入搜索关键字 Selenium
driver.find_element_by_id('sb_form_q').send_keys('Selenium')
driver.quit()
```

下面是一个多层级的嵌套页面在自动化测试中的应用实例，如图 2-3-9 所示。

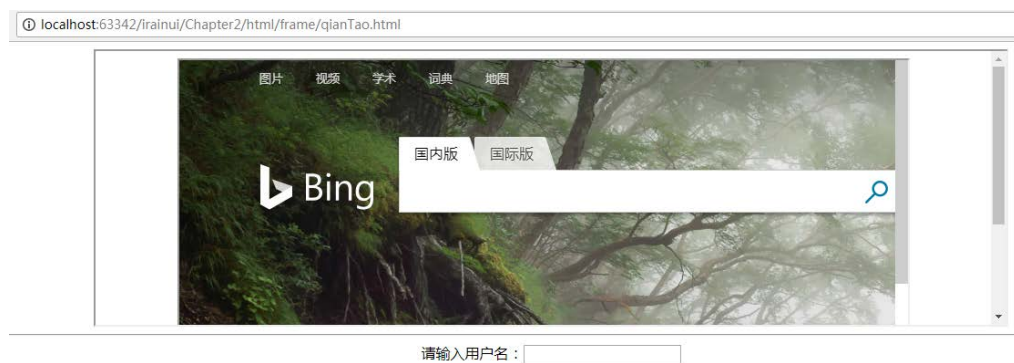


图 2-3-9

HTML 代码如下：

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8" />
<title>多层嵌套</title>
</head>
<body>
```

```

<center>
    <iframe id="parent" src="bing.html" width="1000" height="300"
align="center">

    </iframe>
</center>
<hr>
<center>
    请输入用户名: <input type="text" id="userid" name="username"
class="classname" >
</center>

</body>
</html>

```

在以上 HTML 代码中，可以看到 Bing 首页的搜索多了二层嵌套，页面层级结构如图 2-3-10 所示。

```

▼<iframe id="parent" src="bing.html" width="1000" height="300" align="center">
  ▼#document
    ▼<html>
      ▶<head>...</head>
      ▼<body>
        ▼<center>
          ▶<iframe id="son" src="http://www.bing.com" width="800" height="400">...</iframe>
        </center>
        <br>
        <br>
      </body>
    </html>
  "
  "
</iframe>

```

图 2-3-10

要想在 Bing 首页搜索输入框中输入搜索关键字 Selenium，然后在下面的用户名输入框中输入 Selenium，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver

```

```
driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
#second.html 的地址依据自己本地实际地址
driver.get('file:///D:/git/github/book/html/frame/second.html')
#依据 ID 进入到第一层 frame 框架中
driver.switch_to_frame('parent')
#依据 ID 进入到第二层 frame 框架中
driver.switch_to_frame('son')
#输入搜索关键字 Selenium
driver.find_element_by_id('sb_form_q').send_keys('Selenium')
#跳出 frame 框架
driver.switch_to_default_content()
#用户名输入框中输入 Selenium
driver.find_element_by_name('username').send_keys('Selenium')
driver.quit()
```

第3章 Selenium 与页面的交互

3.1 WebDriver 浏览器的属性

WebDriver 提供了很多属性来支持对浏览器的操作，例如，获取测试地址、多窗口的处理、获取浏览器名称等，具体介绍如下。

1. 获取测试的地址

获取测试的地址用到的方法是 `current_url`，也就是获取当前测试的具体 URL。以百度首页为例，获取的地址就是百度首页的地址，实现的测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
print('测试地址为: {0}'.format(driver.current_url))
driver.quit()
```

运行以上代码后获取的就是百度首页的地址。

2. 获取当前页面代码

获取当前测试页面的代码用到的方法是 `page_source`。以百度首页为例，要想获取百度首页的页面代码，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

```

```
#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
print('页面代码如下: {0}'.format(driver.page_source))
driver.quit()
```

3. 获取当前的 Title

获取当前的 Title，即获取当前测试页面的标题。例如，百度首页的 Title 是“百度一下，你就知道”。获取 Title 的测试代码为：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
print('百度首页的 Title 为: {0}'.format(driver.title))
driver.quit()
```

4. 页面的前进和后退

前进用到的方法是 forward，后退用到的方法是 back。以百度首页和 Bing 搜索首页为例，要实现先打开百度首页，再打开 Bing 搜索页，再后退到百度首页，然后再从百度首页返回到 Bing 搜索页。实现的测试代码为：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
```

```

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
t.sleep(2)
driver.get('http://www.bing.com')
t.sleep(2)
#返回到百度
driver.back()
print('当前 URL 为: {0}'.format(driver.current_url))
t.sleep(2)
#前进到bing
driver.forward()
print('当前 URL 为: {0}'.format(driver.current_url))
driver.quit()

```

5. 关闭程序

在 Selenium 中，quit 方法用来退出驱动程序（Driver）并关闭执行的浏览器；而 close 方法用来关闭执行的浏览器，所以关闭程序建议使用 quit 方法。

6. 加载测试地址

在 UI 自动化测试中，打开测试地址用到的方法是 get 方法，它的参数是要打开的测试页面的地址。例如，要测试打开新浪邮箱的地址（<https://mail.sina.com.cn/>），测试代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://mail.sina.com.cn/')
driver.quit()

```

7. 多窗口实战

在新浪邮箱的登录页面（简称新浪登录页面，或登录页面）点击“注册”按钮，如何在打开的注册页面输入框中输入注册信息呢？这里会用到窗口处理的方法。current_window_handle 用来获取当前浏览器的窗口句柄，window_handles 用

来获取浏览器的所有窗口句柄。要实现在新浪登录页面点击注册，在注册页面邮箱地址输入框中输入邮箱地址，再次跳转到登录页面，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://mail.sina.com.cn/')
driver.implicitly_wait(30)
#获取当前窗口句柄
now_handle=driver.current_window_handle
t.sleep(2)
#点击注册链接
driver.find_element_by_link_text('注册').click()
t.sleep(2)
#获取所有窗口句柄
handles=driver.window_handles
#对所有窗口句柄循环处理
for handle in handles:
    #判断 handle 不是当前窗口句柄
    if handle!=now_handle:
        driver.switch_to_window(handle)
        t.sleep(2)
        driver.find_element_by_name('email').send_keys('wuya')
        t.sleep(2)
        # 关闭注册页面
        driver.close()
#切换到登录页面
driver.switch_to_window(now_handle)
t.sleep(3)
#在账号输入框中输入邮箱
driver.find_element_by_id('freename').send_keys('wuya')
t.sleep(4)
driver.quit()
```

8. 浏览器最大化

使浏览器最大化的方法是 `maximize_window`。一般打开浏览器时，界面并不

是最大化的，这对 UI 自动化测试的影响比较大，所以建议在打开浏览器后，调用该方法让浏览器最大化，实现的测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
#浏览器最大化
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.quit()
```

9. 刷新

刷新用到的方法是 `refresh` 方法。在 UI 自动化测试中，某些场景需要用到页面的刷新。例如，打开百度首页后，输入搜索关键字，然后刷新页面查看效果，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.find_element_by_id('kw').send_keys('Selenium')
t.sleep(2)
#对页面进行刷新
driver.refresh()
t.sleep(3)
driver.quit()
```

10. 获取执行的浏览器

获取执行的浏览器名称用到的是 `name` 方法，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
#测试使用的浏览器
print('测试执行的浏览器为: {0}'.format(driver.name))
driver.quit()
```

3.2 WebElement 类的方法

WebElement 类中有很多方法可以应用在 UI 自动化测试中。例如，get_attribute 方法用以获取输入框中的 Value 值，is_enabled 方法用来判断文本是否可编辑。下面具体介绍这些方法在 UI 自动化测试中的应用。

1. 清空

在 Selenium 中，实现清空用到的方法是 clear。例如，在百度搜索输入框中输入搜索关键字 Selenium，然后再调用 clear 方法清空输入的关键字，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
so=driver.find_element_by_id('kw')
#输入搜索的关键字
so.send_keys('Selenium')
```

```
t.sleep(2)
#清空搜索的关键字
so.clear()
t.sleep(2)
driver.quit()
```

2. 获取元素属性值

`get_attribute` 方法可用来获取元素属性的值。在测试中，经常需要获取输入框中的值，或者获取输入框中的提示信息。仍以新浪邮箱的登录页为例，实现的代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<center>
  <div>
    <p>
      用户名:<input type="text" name="username" placeholder="请输入
用户名">
    </p>
    <p>
      密码: <input type="password" placeholder="请输入密码">
    </p>
  </div>
</center>
</body>
</html>
```

如图 3-2-1 所示。

用户名:

密码:

图 3-2-1

接下来测试用户名输入框是否显示“请输入用户名”的提示信息。“请输入用户名”的元素属性是 `placeholder`，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('file:///D:/git/Python/ActualCombat/Chapter2/html/attribute.html')
name=driver.find_element_by_name('username')
print('用户名输入框中提示信息为：
{0}'.format(name.get_attribute('placeholder'))
t.sleep(2)
driver.quit()
```

在输入框中输入的值一般都在 `Value` 属性中，例如，在百度搜索输入框中输入的搜索关键字存放在百度搜索 `input` 的 `Value` 属性中。要想获取该关键字，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
so=driver.find_element_by_id('kw')
so.send_keys('Selenium WebDriver')
print('百度搜索输入框中填写的关键字为：{0}'.format(so.get_attribute('value')))
driver.quit()
```

注解：运行以上代码后，会显示在搜索框输入的搜索关键字，如图 3-2-2 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter2/webelementTest.py
百度搜索输入框中填写的关键字为:Selenium WebDriver

Process finished with exit code 0
```

图 3-2-2

3. 检查元素是否可见

`is_displayed` 方法用于测试该属性对用户是否可见，返回结果为布尔类型。如果可见，返回的结果为 `True`；如果不可见，返回的结果为 `False`。这里以百度首页为例，测试“关于百度”是否可见，实现的测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('https://www.baidu.com/')
about=driver.find_element_by_link_text('关于百度')
print('关于百度是否可见: {0}'.format(about.is_displayed()))
driver.quit()
```

注解：由于“关于百度”在页面上是可见的，那么运行代码后返回的结果应该是 `True`，如图 3-2-3 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter2/webelementTest.py
关于百度是否可见: True

Process finished with exit code 0
```

图 3-2-3

4. 检查元素是否可编辑

`is_enabled` 方法用于测试文本是否可编辑，返回的结果是布尔类型。如果可

编辑，返回的结果是 `True`；如果不可编辑，返回的结果是 `False`。以百度搜索输入框为例，因为它是可编辑的，所以返回的结果是 `True`，实现的测试代码如下：

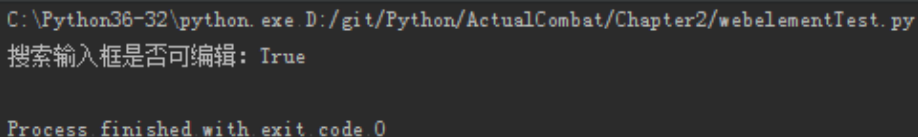
```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('https://www.baidu.com/')
so=driver.find_element_by_id('kw')
print('搜索输入框是否可编辑: {0}'.format(so.is_enabled()))
driver.quit()
```

执行后的结果如图 3-2-4 所示。



```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter2/webelementTest.py
搜索输入框是否可编辑: True

Process finished with exit code 0
```

图 3-2-4

5. 是否已选中

`is_selected` 方法主要针对单选按钮或者复选按钮，判断它们是否已被选中，返回结果是布尔类型。如果选中，返回的结果是 `True`；如果未选中，返回的结果是 `False`。这里以新浪邮箱的登录为例，验证登录页面是否默认选中了自动登录，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
```

```

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://mail.sina.com.cn/')
autoLogin=driver.find_element_by_id('store1')
print('新浪登录页面自动登录是否已默认选中:
{0}'.format(autoLogin.is_selected()))
driver.quit()

```

注解：新浪登录页面的自动登录已默认选中，所以运行以上代码成功后，会返回 True，如图 3-2-5 所示。

```

C:\Python36-32\python.exe D:/git/github/UI/f17.py
新浪登录页面自动登录是否已默认选中: True

Process finished with exit code 0
|

```

图 3-2-5

6. 提交表单

提交表单用到的方法是 submit。例如，在输入框中输入“昵称”，点击“提交”按钮，跳转到另外一个页面，Form 表单的 HTML 代码如下：

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<center>
    <form action="index.html" method="get">
        <p>
            昵称: <input type="text" name="username">
        </p>
        <p>
            <input type="submit" name="form" value="提交">
        </p>
    </form>
</center>

```



```
</body>
</html>
```

点击“提交”按钮后跳转到 index.html 页面，代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<center>
    <h1>
        Hello WuYa!
    </h1>
</center>
</body>
</html>
```

实现以上交互的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('file:///D:/git/Python/ActualCombat/Chapter2/html/submit.html')
driver.find_element_by_name('username').send_keys('WuYa')
driver.find_element_by_name('form').submit()
t.sleep(2)
driver.quit()
```

3.3 下拉框实战

1. Select 类的详解

在 UI 自动化测试中，经常会遇到下拉框的应用。针对下拉框，Selenium 提

供了 `Select` 类来处理, `Select` 类在 `select` 模块中。使用 `Select` 类首先需要导入, 导入方式是 `from selenium.webdriver.support.select import Select`。在 `Select` 类中, 构造方法的参数是 `webelement`, 检查指定的元素时, 如果参数错误就会抛出 `UnexpectedTagNameException` 的异常错误信息。在 `Select` 类中提供了很多方法可在下拉框定位中使用, 下面具体介绍这些方法的应用。

2. 下拉框定位的思路

这里以在百度搜索设置中设定期望搜索结果显示的条数为例, 说明下拉框操作方式, 如图 3-3-1 所示。



图 3-3-1

这部分的 HTML 代码如下:

```
<select id="nr" name="NR">
  <option value="10" selected="">每页显示 10 条</option>
  <option value="20">每页显示 20 条</option>
  <option value="30">每页显示 50 条</option>
</select>
```

下拉框的元素属性 ID 是 `nr`, 在图 3-3-1 中, 实现选择下拉框选项内容的步骤为:

- (1) 首先定位到 `Select` 下拉框的元素属性, 具体代码是 `nr=driver.find_element_by_id('nr')`。
- (2) 实例化 `Select` 类, 参数为 `nr`, 具体代码为 `select=Select(nr)`。
- (3) `Select` 实例化后的对象 `select` 可以调用 `Select` 类的任何一个方法。

3. 索引定位实战

使用索引的方式实现下拉框选项的选择，用到的方法是 `select_by_index`。该方法的参数是索引值。例如，想要实现每页显示 50 条，它在第三位（参见图 3-3-1），由于索引是从 0 开始的，所以索引就是 2，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.select import Select
from selenium.webdriver.common.action_chains import ActionChains
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#实现鼠标悬浮到百度首页的设置
element=driver.find_element_by_css_selector('a.pf:nth-child(8)')
t.sleep(3)
ActionChains(driver).move_to_element(element).perform()
t.sleep(3)
#点击设置中的搜索设置按钮
driver.find_element_by_css_selector('.setpref').click()
t.sleep(2)
#定位到下拉框的元素属性
nr=driver.find_element_by_name('NR')
#实例化 Select 类
select=Select(nr)
select.select_by_index(2)
print ('下拉框选择的最新条数是:',nr.get_attribute('value'))
t.sleep(3)
driver.quit()
```

4. value 定位实战

通过 Value 值可定位下拉框中的选项，用到的方法是 `select_by_value`。该方法的参数是下拉框中具体的 Value 值，需要选择下拉框中的哪个值，直接在该方法的参数中填写具体的 Value 值即可。本例中的 Value 值是 10、20、50，这里选

择在下拉框中每页显示 20 条，则对应的 Value 值是 20，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.select import Select
from selenium.webdriver.common.action_chains import ActionChains
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#实现鼠标悬浮到百度首页的设置
element=driver.find_element_by_css_selector('a.pf:nth-child(8)')
t.sleep(3)
ActionChains(driver).move_to_element(element).perform()
t.sleep(3)
#点击设置中的搜索设置按钮
driver.find_element_by_css_selector('.setpref').click()
t.sleep(2)
#定位到下拉框的元素属性
nr=driver.find_element_by_name('NR')
#实例化 Select 类
select=Select(nr)
#按 value 值的方式来选择下拉框中的内容
select.select_by_value('20')
print ('下拉框选择的最新条数是:',nr.get_attribute('value'))
t.sleep(3)
driver.quit()
```

5. 文本定位实战

使用文本的方式实现对下拉框中选项的选择，用到的方法是 `select_by_visible_text`，参数是具体的 `text` 值。本例中搜索设置下拉框的文本信息具体为“每页显示 10 条”“每页显示 20 条”“每页显示 50 条”。这里选择在下拉框中每页显示 50 条，那么调用的时候，实际参数就是“每页显示 50 条”，实现该测试点的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.select import Select
from selenium.webdriver.common.action_chains import ActionChains
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#实现鼠标悬浮到百度首页的设置
element=driver.find_element_by_css_selector('a.pf:nth-child(8)')
t.sleep(3)
ActionChains(driver).move_to_element(element).perform()
t.sleep(3)
#点击设置中的搜索设置按钮
driver.find_element_by_css_selector('.setpref').click()
t.sleep(2)
#定位到下拉框的元素属性
nr=driver.find_element_by_name('NR')
#实例化 Select 类
select=Select(nr)
#按 value 值的方式来选择下拉框中的内容
select.select_by_visible_text('每页显示 50 条')
print ('下拉框选择的最新条数是:',nr.get_attribute('value'))
t.sleep(3)
driver.quit()
```

3.4 弹出框实战

1. Alert 类的详解

在 UI 自动化测试中经常会遇到 Alert 弹出框的场景。在 HTML 的 Javascript 交互中主要有 Alert 警告框、Confirm 确认框、Prompt 消息对话框。在 Selenium 中对于弹出框的处理方可以使用 alert 模块中的 Alert 类，在 Alert 类中，方法 text 用来获取 Alert 弹出框的文本信息。accept 和 dismiss 方法主要应用在 Confirm 弹出确认框中，accept 用来接受确认框，dismiss 用来拒绝确认框。send_keys 主要应用

在 Prompt 消息对话框中，在输入框中输入要输入的值。在 Selenium 中是通过 switch_to_alert 方法调用 Alert 类中的方法，而 switch_to_alert 方法是属于 WebDriver 类的方法。

2. 警告框的处理

这里以百度搜索设置为例，在进行搜索设置后点击“保存设置”按钮，弹出 alert 对话框，点击“确定”按钮，如图 3-4-1 所示。



图 3-4-1

要想点击“保存设置”按钮后，获取 Alert 弹出框的文本信息，点击弹出框中的“确定”按钮，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.common.alert import Alert
from selenium.webdriver.common.action_chains import ActionChains
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#实现鼠标悬浮到百度首页的设置
element=driver.find_element_by_css_selector('a.pf:nth-child(8)')
t.sleep(3)
```

```

ActionChains(driver).move_to_element(element).perform()
t.sleep(3)
#点击设置中的搜索设置按钮
driver.find_element_by_css_selector('.setpref').click()
t.sleep(2)
#点击搜索设置按钮
driver.find_element_by_css_selector('#gxszButton >
a.prefpanelgo').click()
t.sleep(2)
#获取弹出框的文本信息
print('alert 弹出框的文本信息为: {0}'.format(driver.switch_to_alert().text))
#点击 alert 弹出框中的确定按钮
driver.switch_to_alert().accept()

```

运行以上代码后，会打印出 Alert 的文本信息。

3. 确认框的处理

Confirm 是弹出确认的对话框，用户可以选择“确定”或者“取消”按钮。点击“确定”按钮调用的方法是 `accept`，点击“取消”按钮调用的方法是 `dismiss`，如图 3-4-2 所示。

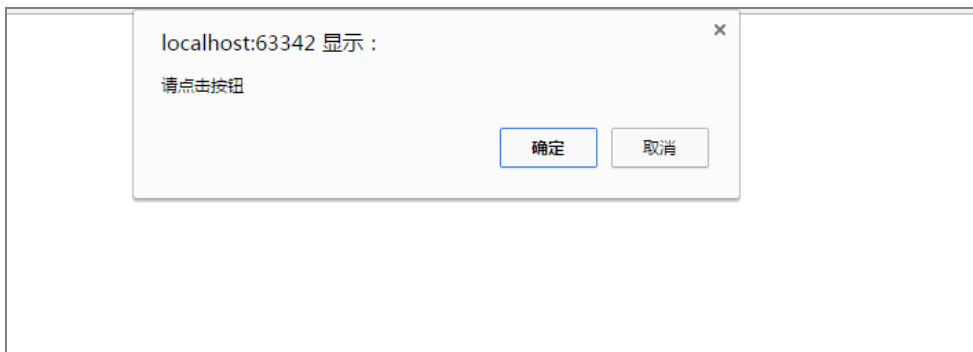


图 3-4-2

这部分的 HTML 交互代码如下：

```

<html>
<meta charset="UTF-8">
<head>
<script type="text/javascript">
    function disp_confirm()
    {
        var r=confirm("请点击按钮")
        if (r==true)

```

```

        {
            document.write("您点击了确认按钮!")
        }
    else
    {
        document.write("您点击了取消按钮!")
    }
}
</script>
</head>
<body>
<center>
    <input type="button" onclick="disp_confirm()" value="请点击我，谢谢！"
/>
</center>
</body>
</html>

```

实现点击“确定”按钮的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.common.alert import Alert
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('file:///D:/git/Python/ActualCombat/Chapter2/html/confirm.html')
#点击按钮
driver.find_element_by_css_selector('body > center > input[type="button"]').click()
t.sleep(2)
#点击“确定”按钮
driver.switch_to_alert().accept()
t.sleep(2)
driver.quit()

```


4. 消息对话框的处理

Prompt 消息对话框，在 JavaScript 中用于询问一些需要与用户交互的信息。要想在 Prompt 消息对话框中输入信息，用到的方法是 `send_keys`，如图 3-4-3 所示。

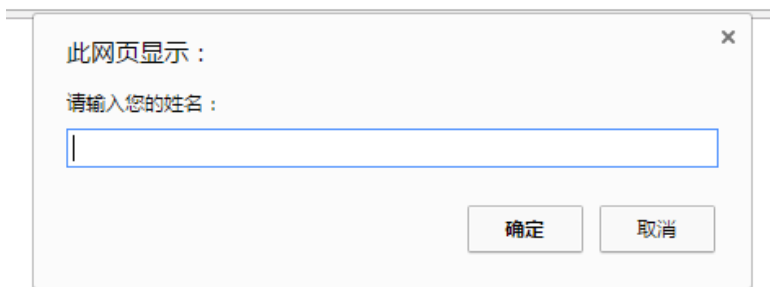


图 3-4-3

实现的 HTML 代码如下：

```
<html>
<meta charset="UTF-8">
<head>
<script type="text/javascript">
    function disp_prompt()
    {
        var name=prompt("请输入您的姓名：","")
        if (name!=null && name!="")
        {
            document.write("Hello " + name + "！")
        }
    }
</script>
</head>
<body>
<center>
    <input type="button" onclick="disp_prompt()" value="请点击我，谢谢！" />
</center>
</body>
</html>
```

在弹出的输入框中输入姓名后，点击“确定”按钮。实现的测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*
```

```

#author:wuya

from selenium import webdriver
from selenium.webdriver.common.alert import Alert
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('file:///D:/git/Python/ActualCombat/Chapter2/html/prompt.html')
#点击按钮
driver.find_element_by_css_selector('body > center >
input[type="button"]').click()
t.sleep(2)
#在弹出的输入框中输入姓名
driver.switch_to_alert().send_keys('无涯')
t.sleep(2)
driver.switch_to_alert().accept()
t.sleep(3)
driver.quit()

```

3.5 WebDriverWait 类实战

在 UI 自动化测试中，首先要保障测试脚本的稳定运行。但是在实际的测试场景中，由于网络的因素导致需要测试的页面打不开或者打开延迟，从而导致页面元素找不到等各种错误的出现。

在 UI 自动化测试中，等待主要存在如下三种形式，分别是：

- (1) 固定等待。如调用 `time` 模块中的 `sleep` 方法，固定等待几秒。
- (2) 隐式等待。用到的方法是 `implicitly_wait`，隐藏式等待指设置最长等待时间。
- (3) 显式等待。主要指程序会每隔一段时间执行自定义的程序判断条件，如果判断条件成功，程序就继续执行；如果判断条件失败，程序就会报 `TimeoutException` 的异常信息。

例如，测试百度首页的搜索时，设置固定等待几秒的时间，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

```

```
#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
t.sleep(3)
driver.find_element_by_id('kw').send_keys('Selenium')
driver.quit()
```

在以上测试代码中，增加了固定等待 3 秒的时间，试想，如果每个步骤都增加等待 3 秒的时间，那么一个测试用例执行下来后，等待的时间就有十几秒，这样会大大地延长测试用例执行的时间。因此，在 Selenium 中提供了 WebDriverWait 类，专门解决网络延迟等待等问题。

1. WebDriverWait 类详解

在学习 WebDriverWait 类应用之前，先来了解一下 WebDriverWait 类的相关概念。WebDriverWait 类放置在 wait 模块下，使用该类的时候，首先需要导入并且对它进行实例化，导入的代码如下：

```
from selenium.webdriver.support.ui import WebDriverWait
```

WebDriverWait 类的参数分别是 driver 和 timeout。driver 指的是 webdriver 实例化后的对象，timeout 指的是具体等待的时间。在 WebDriverWait 类的 until 方法中，参数 method 指的是需要调用的方法，调用的方法来自 expected_conditions 模块中的类，也就是说 WebDriverWait 会与 expected_conditions 模块结合起来应用。调用 expected_conditions 模块中的类之后，一般会返回两种形式的对象实例：一种是布尔类型，返回 True 程序继续执行，如果不是 True，程序就会报 TimeOutException 的异常信息；另外一种返回某个类的具体实例对象，使用该对象就可以直接调用该类中的方法。下面具体看一下该模块中的类在 UI 自动化测试中的应用。

2. 元素可见并且可操作

element_to_be_clickable 判断某元素可见后执行输入、点击等操作。

`element_to_be_clickable` 满足条件后返回的是 `WebElement` 类的实例对象，这样就可以调用 `WebElement` 类中的 `send_keys` 等方法。这里以百度搜索输入框为例，打开百度首页后首先需要加载搜索的输入框，然后才可以在搜索输入框中输入搜索的关键字，否则就返回错误的信息。

实现点击百度首页的“百度一下”按钮的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya


from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
so=WebDriverWait(driver,10).until(expected_conditions.element_to_be_clickable((By.ID,'kw'))
so.send_keys('Selenium')
driver.quit()
```

注解：在以上代码中，我们可以看到 `WebDriverWait` 直接调用静态方法 `until`，并且显式等待 10 s 的时间，在 `until` 的方法中又调用了 `expected_conditions` 模块中的 `element_to_be_clickable` 类，在 `element_to_be_clickable` 类中需要指定按 ID 或者其他元素属性来指定元素的属性。

运行以上代码后，在 10 s 范围内，只要百度首页的搜索输入框可以正确地加载出来，都是可以在输入框中输入搜索的关键字。对以上代码进行修改，故意写错元素属性，把 `kw` 修改成 `kwkw`，执行后，就会提示 `TimeoutException` 的错误提示，如图 3-5-1 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter2/waitInfo.py
Traceback (most recent call last):
  File "D:/git/Python/ActualCombat/Chapter2/waitInfo.py", line 17, in <module>
    so=WebDriverWait(driver,10).until(expected_conditions.element_to_be_clickable((By.ID,'kwkw')))
  File "C:\Python36-32\lib\site-packages\selenium\webdriver\support\wait.py", line 80, in until
    raise TimeoutException(message, screen, stacktrace)
selenium.common.exceptions.TimeoutException: Message:
```

图 3-5-1

3. 指定元素的文本位置

`text_to_be_present_in_element` 指定元素的文本位置，一般用于验证一个文本信息或者错误的提示信息。在这里以新浪邮箱登录页面为例，当账号和密码为空，点击“登录”按钮，验证页面返回的错误提示信息是否为“请输入邮箱名”，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('https://mail.sina.com.cn/#')
#输入新浪邮箱账号
driver.find_element_by_id('freename').send_keys('')
#输入新浪邮箱密码
driver.find_element_by_id('freepassword').send_keys('')
#点击新浪邮箱登录按钮
driver.find_element_by_link_text('登录').click()
WebDriverWait(driver,10).until(expected_conditions.text_to_be_present_in_element((By.XPATH,'/html/body/div[3]/div/div[2]/div/div/div[4]/div[1]/d
```

```
iv[1]/div[1]/span[1]'),'请输入邮箱名'))  
driver.quit()
```

注解：在以上代码中，如果错误提示信息是“请输入邮箱名”，程序执行正确；如果不是，就会提示对应的错误信息。`text_to_be_present_in_element` 返回的结果是 `True` 或者 `False`，当返回结果是 `True` 的时候，执行结果是正确的；当返回的结果是 `False` 的时候，则会提示 `TimeOutExpection` 的错误信息。

因为 `text_to_be_present_in_element` 返回的结果是 `True` 或者 `False`，对以上代码进行修改，见打印的结果：

```
#!/usr/bin/env python  
#-*-coding:utf-8-*-  
  
#author:wuya  
  
from selenium import webdriver  
from selenium.webdriver.support.ui import WebDriverWait  
from selenium.webdriver.support import expected_conditions  
from selenium.webdriver.common.by import By  
import time as t  
  
driver=webdriver.Firefox()  
driver.maximize_window()  
driver.implicitly_wait(30)  
driver.get('https://mail.sina.com.cn/#')  
#输入新浪邮箱账号  
driver.find_element_by_id('freename').send_keys('')  
#输入新浪邮箱密码  
driver.find_element_by_id('freepassword').send_keys('')  
#点击新浪邮箱登录按钮  
driver.find_element_by_link_text('登录').click()  
isText=WebDriverWait(driver,10).until(expected_conditions.text_to_be_present_in_element((By.XPATH,'/html/body/div[3]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]'),'请输入邮箱名'))  
print('打印结果是否是 True:{0}'.format(isText))  
driver.quit()
```

执行后的结果如图 3-5-2 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter2/waitInfo.py
打印结果是否是True:True
```

```
Process finished with exit code 0
```

图 3-5-2

4. 判断元素是否可见

visibility_of_element_located 方法的作用是等元素可见后再执行操作。这里以百度首页的“关于百度”为案例，如果“关于百度”可见，再点击百度首页的“关于百度”链接。实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions
from selenium.webdriver.common.by import By
import time as t

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
aboutBaidu=WebDriverWait(driver,10).until(expected_conditions.visibility_of_element_located((By.LINK_TEXT,'关于百度'))
aboutBaidu.click()
driver.quit()
```

3.6 ActionChains 类实战

在功能的自动化测试中，经常会用到鼠标事件。在 Selenium 中，主要在 action_chains 模块的 ActionChains 类中，导入的方式为：

```
from selenium.webdriver.common.action_chains import ActionChains
```

1. ActionChains 类详解

在 ActionChains 类中提供了对常用鼠标操作的方法，例如，鼠标悬浮到某一元素等操作。要使用 ActionChains 类中的方法首先需要对 ActionChains 类进行实例化，该类的构造函数参数为 driver，也就是 WebDriver 实例化后的对象，对 ActionChinas 类实例化后可以调用它里面的方法，实例化该类的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
from selenium.webdriver.common.action_chains import ActionChains

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')
#对 ActionChains 类进行实例化
actionChains=ActionChains(driver)
driver.quit()
```

对 ActionChains 类实例化后，输入对象 actionChains，按下鼠标，就会显示该类中的方法，如图 3-6-1 所示。


```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# author: wuya

from selenium import webdriver
import time as t
from selenium.webdriver.common.action_chains import ActionChains

driver=webdriver.Firefox()
driver.maximize_window()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com/')
#对ActionChains类进行实例化
actionChains=ActionChains(driver)
actionChains.
driver.quit()
```

click(self, on_element)	ActionChains
click_and_hold(self, on_element)	ActionChains
context_click(self, on_element)	ActionChains
double_click(self, on_element)	ActionChains
drag_and_drop(self, source, target)	ActionChains
drag_and_drop_by_offset(self, source, xoff...	ActionChains
key_down(self, value, element)	ActionChains
key_up(self, value, element)	ActionChains
move_by_offset(self, xoffset, yoffset)	ActionChains
move_to_element(self, to_element)	ActionChains
move_to_element_with_offset(self, to_element...	ActionChains

Press Ctrl+点 to choose the selected (or first) suggestion and insert a dot afterwards >>

图 3-6-1

2. 鼠标悬浮操作

`move_to_element` 方法用来使鼠标悬浮到某一元素上。这里以百度首页的搜索设置为例，要想选择“搜索设置”选项，首先需要鼠标悬浮到“设置”选项上，显示“搜索设置”后才可以点击，如图 3-6-2 所示。

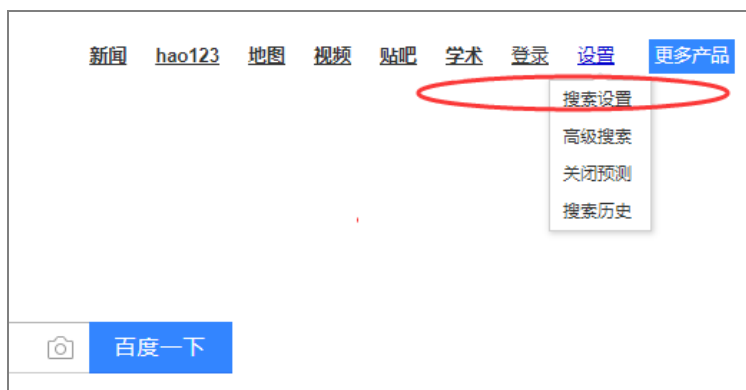


图 3-6-2

要想实现拖动鼠标悬浮到“设置”，再点击“搜索设置”按钮，测试代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
from selenium.webdriver.common.action_chains import ActionChains

driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#对 ActionChains 类进行实例化
actionChains=ActionChains(driver)
locator=driver.find_element_by_css_selector('#ul > a.pf')
#鼠标悬浮到设置
actionChains.move_to_element(locator).perform()
#点击搜索设置
driver.find_element_by_xpath('//*[@id="wrapper"]/div[6]/a[1]').click()
driver.quit()
```

运行以上代码就可以看到，鼠标移动到“设置”选项，显示“搜索设置”选项后，点击“搜索设置”按钮，并跳转到“搜索设置”页面（参见图 3-6-2）。

3. 鼠标右键操作

`content_click` 是触发鼠标右键操作的方法。这里以百度首页为例，要想用鼠标在搜索输入框上点击右键时，弹出鼠标右键的交互信息，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
from selenium.webdriver.common.action_chains import ActionChains
```

```

driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#对 ActionChains 类进行实例化
actionChains=ActionChains(driver)
so=driver.find_element_by_id('kw')
#鼠标在百度搜索输入框右键操作
actionChains.context_click(so).perform()
t.sleep(6)
driver.quit()

```

4. 鼠标双击操作

`double_click` 是触发鼠标双击操作的方法，一般使用在有数据交互的地方。例如，有这样一个业务，点击按钮会向数据库中插入一条数据，而且只能插入一条数据，如果前端对双击没有进行处理的话，双击按钮后就会向数据库插入两条数据。这里以百度首页的“百度一下”按钮为例，演示双击事件在自动化测试中的应用。实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t
from selenium.webdriver.common.action_chains import ActionChains

driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

#对 ActionChains 类进行实例化
actionChains=ActionChains(driver)
driver.find_element_by_id('kw').send_keys('Selenium')
locator=driver.find_element_by_id('su')
#对“百度一下”按钮双击操作

```

```
actionChains.double_click(locator).perform()  
driver.quit()
```

3.7 键盘事件实战

对于键盘事件的操作，Selenium 提供了 keys 模块中的 Keys 类来处理，导入的代码为 `from selenium.webdriver.common.keys import Keys`。这里以百度搜索输入框为例，在搜索输入框中输入搜索的关键字，通过 **Ctrl+A** 全选之后，通过 **Ctrl+C** 复制。按下“Backspace”键删除输入的搜索关键字，然后打开 Bing 首页，把复制的搜索关键字粘贴到 Bing 搜索输入框中，实现的代码如下：

```
#!/usr/bin/env python  
#-*-coding:utf-8-*-  
  
#author:wuya  
  
from selenium import webdriver  
import time as t  
from selenium.webdriver.common.keys import Keys  
  
driver=webdriver.Firefox()  
driver.implicitly_wait(30)  
driver.get('http://www.baidu.com')  
so=driver.find_element_by_id('kw')  
so.send_keys('Selenium')  
#选中输入框的搜索关键字  
so.send_keys(Keys.CONTROL,'a')  
#复制搜索的关键字  
so.send_keys(Keys.CONTROL,'c')  
#按下 Backspace 删除输入的搜索关键字  
so.send_keys(Keys.BACKSPACE)  
#打开 bing 搜索的首页  
driver.get('http://www.bing.com')  
bingSo=driver.find_element_by_id('sb_form_q')  
#复制搜索关键字到 bing 搜索的输入框中  
bingSo.send_keys(Keys.CONTROL,'v')  
driver.quit()
```

3.8 JavaScript 的处理

Selenium 也提供了对 JavaScript 的处理，例如，滑动到浏览器的底部或者顶端，对时间控件及对富文本等的处理，都需要 JavaScript 的脚本配合。在 Selenium 中对 JavaScript 脚本调用的方法是 `execute_script`，下面结合具体的实例说明这部分在自动化测试中的应用。

1. 浏览器滑动操作

在自动化测试中，特别是在数据查询的页面中，由于页面内容太多，经常需要点击浏览器的底部，但往往由于看不到底部导致找不到页面元素而失败。例如，在百度搜索中，搜索后想要前往下一页，需要滑动到浏览器底部才可以找到翻页按钮，然后点击。下面就以百度搜索为例，实现滑动到浏览器的底部或者顶部，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('http://www.baidu.com')

driver.find_element_by_id('kw').send_keys('Selenium')
driver.find_element_by_id('su').click()
#浏览器滑动到底部 js 代码
down="var q=document.documentElement.scrollTop=10000"
t.sleep(3)
#操作 js 实现鼠标滑动到浏览器底部
driver.execute_script(down)
t.sleep(3)
#点击下一页
driver.find_element_by_link_text('下一页>').click()
t.sleep(3)
#浏览器滑动到顶部 js 代码
```

```
up="var q=document.documentElement.scrollTop=0"
#先滑动到底部
driver.execute_script(down)
t.sleep(3)
#操作 js 实现鼠标滑动到浏览器顶部
driver.execute_script(up)
t.sleep(4)
driver.quit()
```

2. 富文本的处理

在很多的 Web 产品中，经常存在富文本的内容，特别是要求在富文本中写入内容。对于富文本的操作不能按照常规的元素定位方式去定位。富文本一般都在 iframe 框架中，所以需要通过 iframe 的 ID 或者索引的方式进入到 iframe 中，再通过 JavaScript 的方式在富文本中输入需要的内容。例如，要想在微信公众号中编写文章，那么首先需要进入到 iframe。微信公众号富文本部分的 HTML 代码如图 3-8-1 所示。

```
<iframe id="ueditor_0" allowtransparency="true" width="100%" height="100%"
frameborder="0" src="javascript:void(function(){document.open();
document.write("<!DOCTYPE html><html xmlns='http://www.w3.org/1999/xhtml' >
<head><style type='text/css'>body{font-family:sans-serif;}</style><link
rel='stylesheet' type='text/css' href='https://res.wx.qq.com/mpres/zh_CN/
htmledition/comm_htmledition/style/widget/ueditor_new/themes/
iframe441a22.css'></head><body class='view' lang='en' ></body><script
type='text/javascript' id='_initialScript'>setTimeout(function()
{window.parent.UE.instants['_ueditorInstant0']._setup(document);},0);var
_tmpScript = document.getElementById('_initialScript');
_tmpScript.parentNode.removeChild(_tmpScript);</script></html>");
document.close();})();" style="height: 400px;">
  >#document
</iframe>
```

图 3-8-1

从图 3-8-1 中可以看到，iframe 的 ID 为 ueditor_0，那么如果想要在该富文本里面写入内容，可单独写一个可调用函数，参数是编写的内容，代码如下：

```
def richText(content):
    '''往富文本里面写入内容'''
    js="document.getElementById('ueditor_0').contentWindow." \
        "document.body.innerHTML='{0}'".format(content)
    driver.execute_script(js)
```

下面就以 UEditor 的富文本为例，实现在富文本中输入“Python 自动化测试实战”的文字，UEditor 要测试的页面如图 3-8-2 所示。

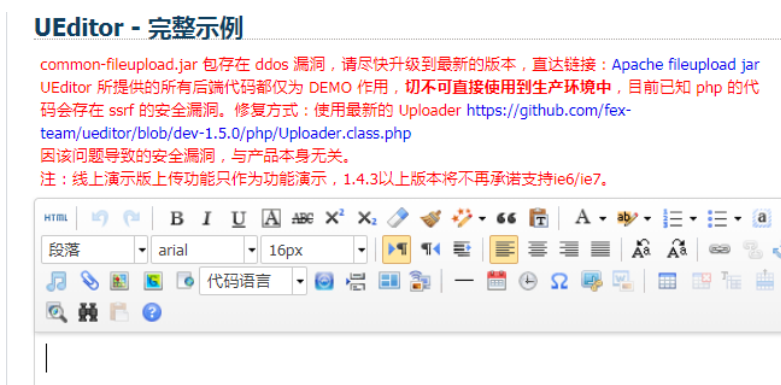


图 3-8-2

实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

def richText(content):
    ''' 往富文本里面写入内容'''
    js="document.getElementById('ueditor_0').contentWindow." \
        "document.body.innerHTML='{0}'".format(content)
    driver.execute_script(js)

driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('http://ueditor.baidu.com/website/onlinedemo.html')
richText('Python 自动化测试实战')
t.sleep(3)
driver.quit()
```

3. 时间控件的处理

时间控件在测试中也是经常遇到的，特别是在查询中，需要按某一时间段查询出数据并且来验证查询功能是否正确。但是，大多数时间控件是只读属性，需

要手动选择时间控件。在自动化测试中怎样自动输入时间呢？时间控件的 UI 交互如图 3-8-3 所示。

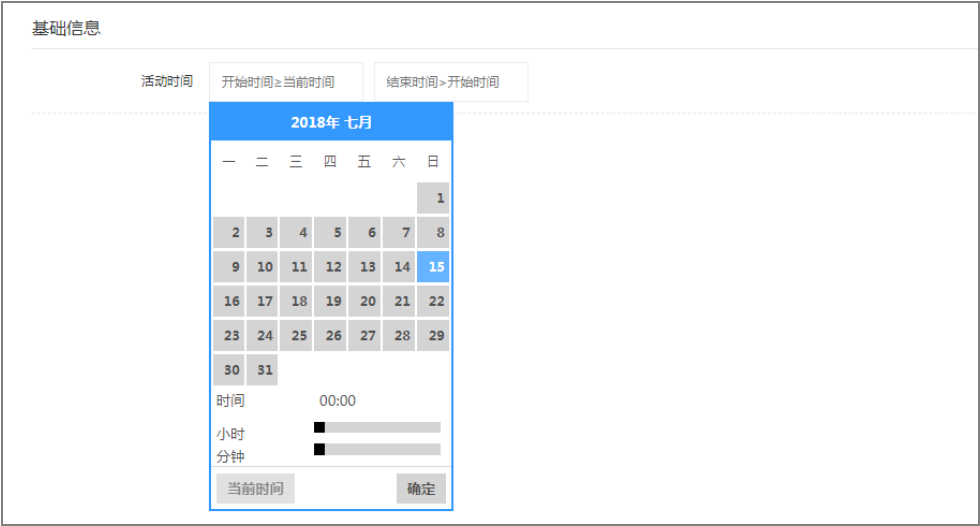


图 3-8-3

这部分的 HTML 代码如图 3-8-4 所示。

```
<div class="wrap">
  <ul class="sub-nav">...</ul>
  <ul class="doc-set">
    <li>
      <div class="doc-dt">...</div>
      <div class="doc-dd">
        <input name="act_start_time" type="text" class="text-box hasDatepicker" value placeholder="开始时间 ≥ 当前时间" title="开始时间 ≥ 当前时间" readonly="readonly" style="cursor:pointer;" id="dp1551339567566"> == $0
        <input name="act_stop_time" type="text" class="text-box hasDatepicker" value placeholder="结束时间 > 开始时间" title="结束时间 > 开始时间" readonly="readonly" style="cursor:pointer;" id="dp1551339567567">
      </div>
    </li>
  </ul>
</div>
```

图 3-8-4

要实现在只读属性的时间控件中输入时间，步骤为：

(1) 取消时间控件的只读属性。

(2) 取消只读属性后，在 Input 输入框中给 Value 赋值，填写需要输入的时间，结合以上步骤，在以上时间控件中，编写开始和结束时间的可调用函数，代码如下：


```

def startJs(startTime):
    ''' 开始时间中输入自定义的时间'''
    js="$("${input[placeholder='开始时间≥当前时间'
']}\").removeAttr('readonly');" \
        "${input[placeholder='开始时间≥当前时间'
']}\").attr('value','{0}').format(startTime)
    driver.execute_script(js)

def endJs(endTime):
    ''' 结束时间中输入自定义的时间'''
    js="$("${input[placeholder='结束时间>开始时间'
']}\").removeAttr('readonly');" \
        "${input[placeholder='结束时间>开始时间'
']}\").attr('value','{0}').format(endTime)
    driver.execute_script(js)

```

要想在打开该页面后，在活动时间中输入开始时间和结束时间，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
import time as t

def startJs(startTime):
    ''' 开始时间中输入自定义的时间'''
    js="$("${input[placeholder='开始时间≥当前时间'
']}\").removeAttr('readonly');" \
        "${input[placeholder='开始时间≥当前时间'
']}\").attr('value','{0}').format(startTime)
    driver.execute_script(js)

def endJs(endTime):
    ''' 结束时间中输入自定义的时间'''
    js="$("${input[placeholder='结束时间>开始时间'
']}\").removeAttr('readonly');" \
        "${input[placeholder='结束时间>开始时间'
']}\").attr('value','{0}').format(endTime)
    driver.execute_script(js)

```

```
driver=webdriver.Firefox()
driver.implicitly_wait(30)
driver.get('file:///C:/Users/Administrator/Desktop/Time/Time/index.html')
startJs('2018-01-01 00:00:00')
t.sleep(3)
endJs('2018-08-08 23:59:59')
driver.quit()
```

运行以上代码后，可看到在活动时间中实现了自定义输入开始时间和结束时间。

3.9 获取截图

在自动化测试中，在测试执行期间获取到截图信息，一方面可以定位错误的脚本，方便调试错误代码；另外一方面也可以以此为据，与开发人员进行有效的沟通。

1. 获取当前截图

`save_screenshot` 是获取当前截图的方法。以百度首页搜索为例，要想打开百度首页后获取百度首页的截图，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
driver.save_screenshot('baidu.png')
driver.quit()
```

代码执行后会在当前目录下生成 `baidu.png` 的图片。

2. 保存当前屏幕快照

`get_screenshot_as_file` 方法可以将当前的屏幕快照保存成 `.png` 文件，保存文

件的时候可以填写完整的文件路径。依然以百度首页为例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
driver.get_screenshot_as_file('c:/baidu.png')
driver.quit()
```

运行代码后会在 C 盘下生成 `baidu.png` 的图片，打开后显示的是百度的首页。

3. 获取图片二进制数据

`get_screenshot_as_png` 方法用来获取截取图片的二进制数据。该方法在实际的测试场景中使用较少。以百度首页为例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver

driver=webdriver.Firefox()
driver.maximize_window()
driver.get('http://www.baidu.com')
driver.implicitly_wait(30)
print( driver.get_screenshot_as_png())
driver.quit()
```

第 4 章 单元测试框架 unittest

4.1 unittest 简述

unittest 是 Python 语言的单元测试框架，在 Python 的官方文档中，对 unittest 单元测试框架进行了详细的介绍，感兴趣的读者可以到 <https://www.python.org/doc/> 网站了解。本章重点介绍 unittest 单元测试框架在自动化测试中的应用。

unittest 单元测试框架提供了创建测试用例、测试套件和批量执行测试用例的方案。在 python 安装成功后，unittest 单元测试框架就可以直接导入使用，它属于标准库。作为单元测试的框架，unittest 单元测试框架也是对程序的最小模块进行的一种敏捷化测试。在自动化测试中，我们虽然不需要做白盒测试，但是必须知道所使用语言的单元测试框架，这是因为当我们把 Selenium2 的 API 全部学习完后，就会遇到用例的组织问题。虽然函数式编程和面向对象编程提供了对代码的重构，但是对于所编写的每个测试用例，不可能编写成一个函数（方法）来调用执行。利用单元测试框架，可以创建一个类，该类继承 unittest 的 TestCase，这样可以把每个 TestCase 看成是一个最小的单元，由测试套件组织起来，运行时直接执行即可，同时可引入测试报告。unittest 各个组件的关系如图 4-1-1 所示。

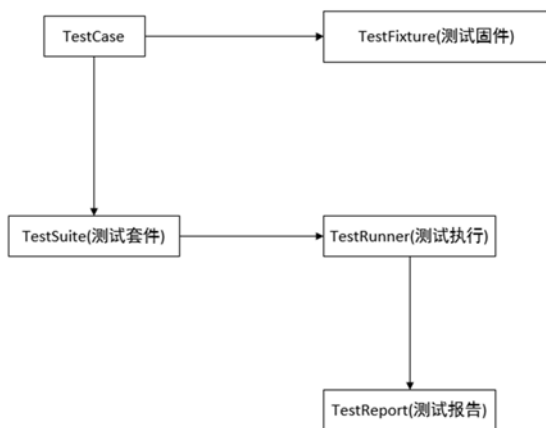


图 4-1-1

4.2 测试固件

在 `unittest` 单元测试框架中，测试固件用于处理初始化的操作，例如，在对百度的搜索进行测试之前，首先需要打开浏览器并且进入到百度首页；测试结束后，需要关闭浏览器。测试固件提供了两种执行形式，一种是每执行一个测试用例，测试固件都会被执行到；另外一种是不管有多少个测试用例，测试固件只执行一次。

1. 测试固件每次均执行

`unittest` 单元测试框架提供了名为 `setUp` 和 `tearDown` 的测试固件。下面，我们通过编写一个例子来看测试固件执行的方式，测试代码为：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        print('start')

    def tearDown(self):
        print('end')

    def test_baidu_so(self):
        print('测试用例执行')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

它的执行顺序是先执行 `setUp` 方法，再执行具体的测试用例 `test_baidu_so`，最后执行 `tearDown` 方法。执行后的结果如图 4-2-1 所示。

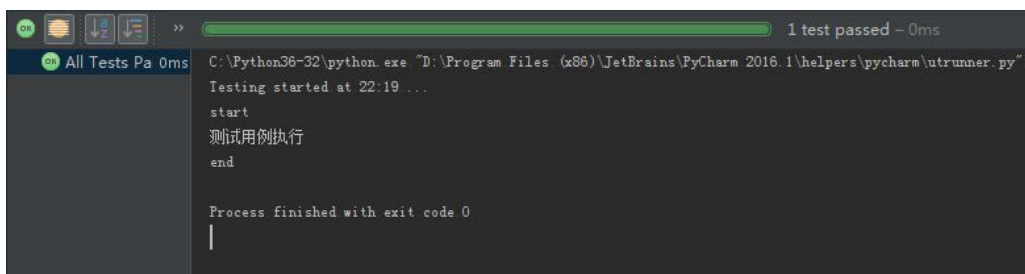


图 4-2-1

下面以百度首页为例，编写两个测试点。执行的方式是 `setUp` 执行两次，`tearDown` 也会执行两次，具体实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_news(self):
        '''验证:测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()

    def test_baidu_map(self):
        '''验证:测试百度首页点击地图后的跳转'''
        self.driver.find_element_by_link_text('地图').click()

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

运行以上代码后，浏览器会被打开两次，也会关闭两次。如果是在一个测试类中有 N 个测试用例，那么也就意味着打开 N 次浏览器，关闭 N 次浏览器，

而关闭和打开浏览器都会占用一定的资源和时间，很显然，这并不是一个理想的选择。

2. 测试固件只执行一次

钩子方法 `setUp` 和 `tearDown` 虽然经常使用，但是在 UI 自动化测试中，一个系统的测试用例一般多达五百多条，打开和关闭五百多次浏览器，会消耗大量的资源和时间。在 `unittest` 单元测试框架中可以使用另外一个测试固件来解决这一问题，它就是 `setUpClass` 和 `tearDownClass` 方法。该测试固件方法是类方法，需要在方法上面加装饰器 `@classmethod`。使用该测试固件，不管有多少个测试用例，测试固件只执行一次，也就是说不管有多少个测试用例，执行的时候浏览器只会被打开一次和关闭一次，案例代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest

class UiTest(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        print('start')

    @classmethod
    def tearDownClass(cls):
        print('end')

    def test_001(self):
        print('第一个测试用例')

    def test_002(self):
        print('第二个测试用例')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

执行以上测试代码后，输出的结果如图 4-2-2 所示。

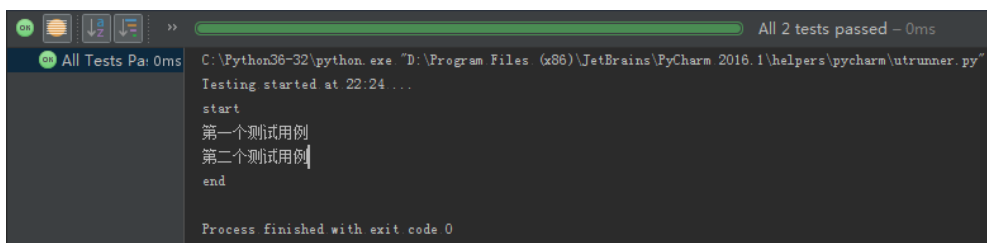


图 4-2-2

注解：在以上代码中，虽然有两个测试方法，但是测试固件只执行了一次。

虽然使用类测试固件可以在执行多个测试用例时让测试固件只执行一次，但是实际使用类测试固件中，还需要解决另外一个问题。例如，以百度首页测试点击新闻页面和测试点击地图页面为例，意味着点击新闻页面后，需回到首页后才可以找得到地图页面的链接进行点击。

因为在新闻页面并没有地图的链接地址，从而导致地图页面的测试失败，反之亦然。实例代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.driver=webdriver.Firefox()
        cls.driver.maximize_window()
        cls.driver.get('http://www.baidu.com')
        cls.driver.implicitly_wait(30)

    @classmethod
    def tearDownClass(cls):
        cls.driver.quit()

    def test_baidu_news(self):
        '''验证:测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        self.driver.get('http://www.baidu.com')

    def test_baidu_map(self):
```



```

'''验证:测试百度首页点击地图后的跳转'''
self.driver.find_element_by_link_text('地图').click()
self.driver.get('http://www.baidu.com')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

运行以上代码后，所有的测试用例执行通过。在实际的自动化测试工作中，建议大家尽量使用测试固件 `setUp` 和 `tearDown`，使得自动化测试用例之间没有关联性，避免一个测试用例执行失败是由上一个测试用例导致。

4.3 测试执行

在以上实例中，可以看到测试用例的执行是在主函数中，`unittest` 调用的是 `main`，代码如下：

```
main = TestProgram
```

`TestProgram` 还是一个类，再来看该类的构造函数，代码如下：

```

class TestProgram(object):
    """A command-line program that runs a set of tests; this is
    primarily
        for making test modules conveniently executable.
    """
    USAGE = USAGE_FROM_MODULE

    # defaults for testing
    failfast = catchbreak = buffer = progName = None

    def __init__(self, module='__main__', defaultTest=None, argv=None,
                  testRunner=None, testLoader=loader.defaultTestLoader,
                  exit=True, verbosity=1, failfast=None,
                  catchbreak=None,
                  buffer=None):
        if isinstance(module, basestring):
            self.module = __import__(module)
            for part in module.split('.')[1:]:
                self.module = getattr(self.module, part)
        else:
            self.module = module
        if argv is None:
            argv = sys.argv

```

```

self.exit = exit
self.failfast = failfast
self.catchbreak = catchbreak
self.verbosity = verbosity
self.buffer = buffer
self.defaultTest = defaultTest
self.testRunner = testRunner
self.testLoader = testLoader
self.progName = os.path.basename(argv[0])
self.parseArgs(argv)
self.runTests()

```

在 `unittest` 模块中包含的 `main` 方法，可以方便地将测试模块转变为可以运行的测试脚本。`main` 使用 `unittest.TestLoader` 类来自动查找和加载模块内的测试用例，`TestProgram` 类中该部分的代码如下：

```

def createTests(self):
    if self.testNames is None:
        self.test = self.testLoader.loadTestsFromModule(self.module)
    else:
        self.test = self.testLoader.loadTestsFromNames(self.testNames,
                                                         self.module)

```

在执行测试用例时，在 `main` 方法中加入了 `verbosity=2`，代码如下：

```

unittest.main(verbosity=2)

```

下面详细地解释一下 `verbosity` 部分。在 `verbosity` 中默认是 1，0 代表执行的测试总数和全局结果，2 代表显示详细的信息。

我们现在通过一个实例来说明执行结果的显示，实例代码如下：

```

#!/usr/bin/env python
#coding:utf-8

#Author:无涯

import unittest
from selenium import webdriver

class Baidu(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Chrome()
        self.driver.maximize_window()

```

```

self.driver.implicitly_wait(30)
self.driver.get('http://www.baidu.com')

def tearDown(self):
    self.driver.quit()

def test_baidu_title(self):
    '''验证百度首页的title'''
    self.assertEqual(self.driver.title, '百度一下，你就知道')

def test_baidu_url(self):
    '''验证百度首页的URL'''
    self.assertEqual(self.driver.current_url, 'https://www.baidu.com')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

执行后的结果如图 4-3-1 所示。

```

test_baidu_title (__main__.Baidu)
验证百度首页的title ... ok
test_baidu_url (__main__.Baidu)
验证百度首页的URL ... FAIL

=====
FAIL: test_baidu_url (__main__.Baidu)
验证百度首页的URL

-----
Traceback (most recent call last):
  File "D:/git/github/UI/tests/test_ui.py", line 25, in test_baidu_url
    self.assertEqual(self.driver.current_url, 'https://www.baidu.com')
AssertionError: 'https://www.baidu.com/' != 'https://www.baidu.com'
- https://www.baidu.com/
? .....
+ https://www.baidu.com

-----

Ran 2 tests in 21.160s

FAILED (failures=1)

```

图 4-3-1

注解：在以上的截图中可以看到，成功的测试用例会显示 **OK**，失败的测试用例会显示出详细的信息。

在一个测试类中，有很多的测试用例，如果想单独地执行某一个测试，用鼠标右键点击要执行的测试用例的名称，选择“Run”。如我们想单独执行 test_baidu_news 的 TestCase，将鼠标移动到该 TestCase 名称上用右键点击，再在出现的菜单项中点击 Run “Unittest test_baidu_news”，如图 4-3-2 所示。

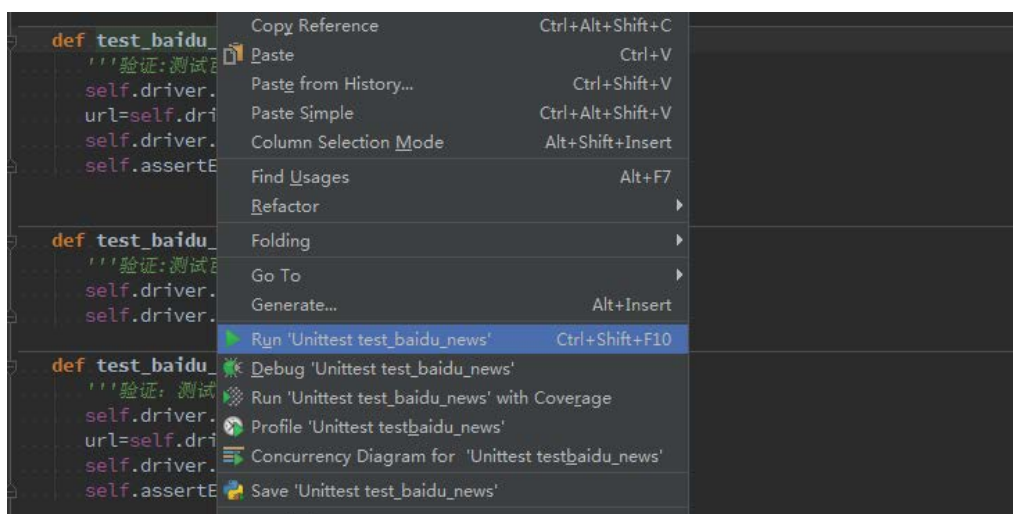


图 4-3-2

点击 Run 后，就会单独地执行 test_baidu_news 的测试用例。

4.4 构建测试套件

前面介绍了测试用例的执行，在一个测试类中会有很多个测试用例。如何来组织并使用这些测试用例呢？unittest 提供了“测试套件”方法，它由 unittest 模块中的 TestSuite 类表示，测试套件可以根据所测试的特性把测试用例组合在一起。

1. 按顺序执行

在实际的工作中，由于业务场景需要测试用例按顺序执行，例如，先执行 X 测试用例再执行 Y 测试用例，在 TestSuite 类中提供了 addTest 方法可以实现，也就是说要执行的测试用例按自己期望的执行顺序添加到测试套件中。下面的案例

实现对百度首页的测试，测试用例的执行顺序是先测试百度新闻，再测试百度地图，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_news(self):
        '''验证: 测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        url=self.driver.current_url
        self.assertEqual(url,'http://news.baidu.com/')

    def test_baidu_map(self):
        '''验证: 测试百度首页点击地图后的跳转'''
        self.driver.find_element_by_link_text('地图').click()
        self.driver.get('http://www.baidu.com')

if __name__ == '__main__':
    suite=unittest.TestSuite()
    suite.addTest(BaiduTest('test_baidu_news'))
    suite.addTest(BaiduTest('test_baidu_map'))
    unittest.TextTestRunner(verbosity=2).run(suite)
```

注解：在以上代码中，首先需要对 `TestSuite` 类进行实例化，使之成为一个对象 `suite`，然后调用 `TestSuite` 类中 `addTest` 方法，把测试用例添加到测试套件中，最后执行测试套件，从而执行测试套件中的测试用例。

运行以上代码后，测试用例会按照添加到测试套件的顺序执行，也就是说先添加进去的先执行，后添加进去的后执行。

2. 按测试类执行

在自动化测试中，一般测试用例往往多达几百个，如果完全按顺序来执行，其一是不符合自动化测试用例的原则，因为在 UI 自动化测试中，自动化测试用例最好独立执行，互相之间不影响并且没有依赖关系。其二是当一个测试类中有很多测试用例时，逐一地向套件中添加用例是一项很烦琐的工作，这时，可以使用 `makeSuite` 类按测试类来执行。`makeSuite` 可以实现把测试用例类中所有的测试用例组成测试套件 `TestSuite` 这样可避免逐一向测试套件中添加测试用例。修改后的测试类中的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_news(self):
        '''验证: 测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        url=self.driver.current_url
        self.assertEqual(url,'http://news.baidu.com/')

    def test_baidu_map(self):
        '''验证: 测试百度首页点击地图后的跳转'''
        self.driver.find_element_by_link_text('地图').click()
        self.driver.get('http://www.baidu.com')
```

```
if __name__ == '__main__':
    suite=unittest.TestSuite(unittest.makeSuite(BaiduTest))
    unittest.TextTestRunner(verbosity=2).run(suite)
```

注解：在以上代码中可以看到，在测试套件 `TestSuite` 类中，`unittest` 模块调用了 `makeSuite` 的方法，`makeSuite` 方法的参数是 `testCaseClass`，也就是测试类，代码如下：

```
def makeSuite(testCaseClass, prefix='test', sortUsing=cmp,
              suiteClass=suite.TestSuite):
    return _makeLoader(prefix, sortUsing,
                       suiteClass).loadTestsFromTestCase(testCaseClass)
```

3. 加载测试类

在 `unittest` 模块中也可以使用 `TestLoader` 类来加载测试类，也就是说 `TestLoader` 加载测试类并将它们返回添加到 `TestSuite` 中，`TestLoader` 类的应用代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_news(self):
        '''验证: 测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        url=self.driver.current_url
```

```

self.assertEqual(url, 'http://news.baidu.com/')

def test_baidu_map(self):
    '''验证: 测试百度首页点击地图后的跳转'''
    self.driver.find_element_by_link_text('地图').click()
    self.driver.get('http://www.baidu.com')

if __name__ == '__main__':
    suite=unittest.TestLoader().loadTestsFromTestCase(BaiduTest)
    unittest.TextTestRunner(verbosity=2).run(suite)

```

4. 按测试模块执行

在 `TestLoader` 类中也可以按模块来执行测试。在 Python 中，一个 Python 文件就是一个模块，一个模块中可以有 N 个测试类，在一个测试类中可以有 N 个测试用例。按模块执行的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.get('http://www.baidu.com')

    def test_title(self):
        '''验证: 测试百度浏览器的title'''
        self.assertEqual(self.driver.title, '百度一下，你就知道')

    def test_so(self):
        '''验证: 测试百度搜索输入框是否可编辑'''
        so=self.driver.find_element_by_id('kw')
        self.assertTrue(so.is_enabled())

    def test_002(self):
        '''验证: 点击百度新闻'''
        self.driver.find_element_by_link_text('新闻').click()

```



```

@unittest.skip('do not run')
def test_003(self):
    '''验证: 点击百度地图'''
    self.driver.find_element_by_link_text('地图').click()

def tearDown(self):
    self.driver.quit()

class BaiduMap(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_map(self):
        '''验证: 测试百度首页点击地图后的跳转'''
        self.driver.find_element_by_link_text('地图').click()
        self.driver.get('http://www.baidu.com')

if __name__ == '__main__':
    suite=unittest.TestLoader().loadTestsFromModule('unittest1.py')
    unittest.TextTestRunner(verbosity=2).run(suite)

```

注解：在以上代码中可以看到，测试类分别是 `BaiduMap` 和 `BaiduTest`，模块名称为 `unittest1.py`，`TestLoader` 类直接调用 `loadTestsFromModule` 方法返回给指定模块中包含的所有测试用例套件。

5. 优化测试套件

以上实例是把测试套件写在了 `main` 主函数中，也可以单独把测试套件写成一个方法来调用。这里以加载测试类为例，把测试套件写成一个单独的方法，代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

```

```

from selenium import webdriver
import unittest

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_baidu_news(self):
        '''验证: 测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        self.assertEqual(self.driver.current_url, 'http://news.baidu.com/')

    def test_baidu_map(self):
        '''验证: 测试百度首页点击地图后的跳转'''
        self.driver.find_element_by_link_text('地图').click()
        self.assertEqual(self.driver.current_url, 'https://map.baidu.com/')

    @staticmethod
    def suite(testCaseClass):
        suite=unittest.TestLoader().loadTestsFromTestCase(testCaseClass)
        return suite

if __name__ == '__main__':
    unittest.TextTestRunner(verbosity=2).run(
        BaiduTest.suite(BaiduTest))

```

4.5 分离测试固件

在 UI 自动化测试中, 不管编写哪个模块的测试用例, 都需要首先在测试类中编写测试固件初始化 `WebDriver` 类及打开浏览器, 测试用例执行完成后还需要关闭浏览器, 这部分的代码如下:

```

def setUp(self):
    self.driver=webdriver.Firefox()

```

```

        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

```

在每一个测试类中都要编写以上代码，因此需要重复编写很多代码。是否可以把测试固件这部分代码分离出去，测试类直接继承分离出去的类呢？我们把测试固件分离到 `init.py` 模块中，类名称为 `InitTest`，代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver

class InitTest(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://www.baidu.com')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

```

测试类继承了 `InitTest`，继承后，在测试类中直接编写要执行的测试用例，代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from init import InitTest

class BaiduTest(InitTest):
    def test_baidu_news(self):
        '''验证: 测试百度首页点击新闻后的跳转'''
        self.driver.find_element_by_link_text('新闻').click()
        self.assertEqual(self.driver.current_url,

```

```

'http://news.baidu.com/')

def test_baidu_map(self):
    '''验证:测试百度首页点击地图后的跳转'''
    self.driver.find_element_by_link_text('地图').click()
    self.assertEqual(self.driver.current_url,
'http://map.baidu.com/')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：首先需要导入 `init` 模块中的 `InitTest` 类，测试类 `BaiduTest` 继承 `InitTest` 类。这样执行测试类后，会先执行 `setUp` 方法，再执行具体的测试用例，最后执行 `tearDown` 方法。`python` 的类继承的方式解决了在每个测试类中都需要编写测试固件的问题。把测试固件分离出去后，即使后期测试地址发生变化，只需要修改 `init` 模块中 `InitTest` 类中的 `url` 地址即可，而不需要在每个测试类修改测试地址，减少了编写重复性代码的开销。

分离了测试固件，运行以上代码，对应的测试用例执行通过。

4.6 测试断言

断言就是判断实际测试结果与预期结果是否一致，一致则测试通过，否则失败。因此，在自动化测试中，无断言的测试用例是无效的。这是因为当一个功能自动化已全部实现，在每次版本迭代中执行测试用例时，执行的结果必须是权威的，也就是说自动化测试用例执行结果应该无功能性或者逻辑性问题。在自动化测试中最忌讳的就是自动化测试的用例功能测试虽然是通过的，但被测功能本身却是存在问题的。自动化测试用例经常应用在回归测试中，发现的问题不是特别多，如果测试结果存在功能上的问题，则投入了人力去做的自动化测试就没有多大的意义了。所以每一个测试用例必须要有断言。在测试的结果中只有两种可能，一种是执行通过，另外一种执行失败，也就是功能存在问题。在 `TestCase` 类中提供了 `assert` 方法来检查和报告失败，常用的方法如图 4-6-1 所示。

Method	Checks that	New in
<code>assertEqual(a, b)</code>	<code>a == b</code>	
<code>assertNotEqual(a, b)</code>	<code>a != b</code>	
<code>assertTrue(x)</code>	<code>bool(x) is True</code>	
<code>assertFalse(x)</code>	<code>bool(x) is False</code>	
<code>assertIs(a, b)</code>	<code>a is b</code>	3.1
<code>assertIsNot(a, b)</code>	<code>a is not b</code>	3.1
<code>assertIsNone(x)</code>	<code>x is None</code>	3.1
<code>assertIsNotNone(x)</code>	<code>x is not None</code>	3.1
<code>assertIn(a, b)</code>	<code>a in b</code>	3.1
<code>assertNotIn(a, b)</code>	<code>a not in b</code>	3.1
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>	3.2
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>	3.2

图 4-6-1

1. assertEquals

`assertEquals` 方法用于测试两个值是否相等，如果不相等则测试失败。在这里特别强调的是，两个值相等不仅仅指内容相同而且还要类型相同。例如，两个值虽然内容一致，但一个是 `bytes` 类型一个是 `str` 类型，则两个值仍为不相等，测试失败。下面通过一个实例来演示，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from selenium import webdriver

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.get('http://www.baidu.com')

    def tearDown(self):
        self.driver.quit()
```

```
def test_baidu_title(self):
    '''验证:测试百度首页的title'''
    self.assertEqual(self.driver.title, '百度一下, 你就知道'.encode('gbk'))

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

以上代码会执行失败，这是因为 `self.driver.title` 获取的内容是“百度一下，你就知道”，它的类型是 `str` 类型；而“百度一下，你就知道”被转为 `bytes` 类型，内容一致、类型不一致，所以失败，执行后的结果如图 4-6-2 所示。

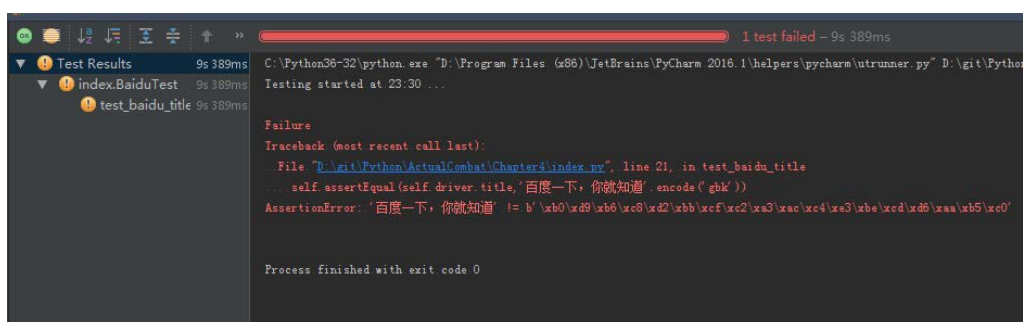


图 4-6-2

我们把代码再次修改一下，让两个断言内容一致，类型也一致，修改后的代码如下：

```
import unittest
from selenium import webdriver

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.implicitly_wait(30)
        self.driver.get('http://www.baidu.com')

    def tearDown(self):
        self.driver.quit()

    def test_baidu_title(self):
        '''验证:测试百度首页的title'''
        self.assertEqual(self.driver.title, '百度一下, 你就知道')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

再次运行以上代码，测试用例执行成功，断言通过。

2. assertTrue

`assertTrue` 返回的是布尔类型，它主要对返回的测试结果执行布尔类型的校验。例如，验证百度搜索输入框是否可编辑，`is_enabled` 方法返回的结果是 `True`，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from init import InitTest

class BaiduTest(InitTest):
    def test_baidu_news(self):
        '''验证: 测试百度搜索输入框是否可编辑'''
        so=self.driver.find_element_by_id('kw')
        self.assertTrue(so.is_enabled())

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

3. assertFalse

`assertFalse` 和 `assertTrue`，都是对返回的布尔类型进行校验。不同的是，`assertFalse` 要求返回的结果是 `False`，测试用例才会执行成功。以新浪邮箱登录页面为例，取消“自动登录”按钮后，方法 `is_elected` 返回的是 `False`，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from init import InitTest

class BaiduTest(InitTest):
    def test_baidu_news(self):
        '''验证: 新浪邮箱登录页面取消自动登录'''
        isautoLogin=self.driver.find_element_by_id('store1')
        isautoLogin.click()
        self.assertFalse(isautoLogin.is_selected())
```

```
if __name__ == '__main__':
    unittest.main(verbosity=2)
```

4. assertIn

assertIn 指的是一个值是否包含在另外一个值的范围内。还是以百度首页的 url 为例，测试 <https://www.baidu.com/> 是否在 <https://www.baidu.com/> 的范围内，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from init import InitTest

class BaiduTest(InitTest):
    def test_baidu_news(self):
        '''验证: 新浪邮箱登录页面取消自动登录'''
        self.assertIn(self.driver.current_url, 'https://www.baidu.com/')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

4.7 断言的注意事项

前面介绍了 unittest 测试框架中断言的基本内容，之所以单独列出一节来讲断言的注意事项，这是因为在实际的工作过程中，以及在与其他的网友交流时，发现很多人的测试用例写得不够规范，导致即使产品的功能有问题，测试用例执行结果仍为正确。出现这种情况，多是因为在测试过程中，使用了不正确的 if 应用和异常应用。下面就从这两个维度来解答这个错误信息是如何发生的，以及怎么样来避免它。

1. 不正确的 if 应用

还是以新浪登录页面为例，来测试自动登录按钮是否选中，该代码中引入了判断的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

```



```

import unittest
from selenium import webdriver

class BaiduTest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_sina_login(self):
        '''验证: 新浪登录页面取消自动登录'''
        isAutoLogin=self.driver.find_element_by_id('store1')
        if isAutoLogin.is_selected():
            print 'success'
        else:
            print 'fail'

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：一个测试用例只有两种结果，要么 Pass，要么 Fail（代码错误也显示 Fail）。以上代码不管复选框是不是自动选中的，测试执行结果都是 Pass，测试用例都是通过。因此，在自动化测试的测试用例中，切记不要使用 if else 这类判断代码来代替断言。

2. 不正确的异常应用

这里还是以新浪登录页面为例，先看以下实例代码：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver

class BaiduTest(unittest.TestCase):

    def setUp(self):
        self.driver = webdriver.Firefox()

```

```

self.driver.maximize_window()
self.driver.get('http://mail.sina.com.cn/')
self.driver.implicitly_wait(30)

def tearDown(self):
    self.driver.quit()

def test_sina_login(self):
    '''验证:新浪登录页面取消自动登录'''
    isAutoLogin=self.driver.find_element_by_id('store1')
    try:
        self.assertTrue(isAutoLogin.is_selected())
    except:
        print 'fail'

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：以上代码和应用 if else 的结果一样，即使自动登录未被选中，测试用例的结果也显示 Pass。另外，在自动化测试中尽量不要应用打印结果来判断测试用例的情况，用例如，果在代码错误或者功能有 Bug 的情况下就让用例报错或者失败，而不是结果显示 Pass，只有功能正常的测试用例结果才是 Pass 的。

4.8 批量执行测试用例

在实际测试中，常常需要批量执行测试用例。例如，在 testCase 包中有 test_baidu.py 和 test_sina.py 两个文件，下面批量执行这两个模块的测试用例。创建新文件 allTests.py，在 allTests.py 文件中编写批量执行的代码，test_baidu.py 模块的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver

class BaiduTest(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()

```

```

self.driver.get('http://www.baidu.com')
self.driver.implicitly_wait(30)

def tearDown(self):
    self.driver.quit()

def test_baidu_title(self):
    '''验证：测试百度首页的title 是否正确'''
    self.assertEqual(self.driver.title, '百度一下，你就知道')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

test_sina.py 模块代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver

class SinaTest(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_username_password_null(self):
        '''验证：新浪登录页面用户名和密码为空错误提示信息'''
        self.driver.find_element_by_id('freename').send_keys('')
        self.driver.find_element_by_id('freepassword').send_keys('')
        self.driver.find_element_by_link_text('登录').click()

divError=self.driver.find_element_by_xpath('/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]').text
self.assertEqual(divError, '请输入邮箱名')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

allTests.py 模块的代码如下:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
import os

def allCases():
    ''' 获取所有测试模块'''
    suite=unittest.TestLoader().discover(
        start_dir= os.path.dirname(__file__),
        pattern='test_*.py',
        top_level_dir=None)
    return suite

if __name__ == '__main__':
    unittest.TextTestRunner(verbosity=2).run(allCases())
```

注解: 在以上代码中, 批量获取测试模块用到的方法是 discover。discover 方法有三个参数, 第一个参数 start_dir 是测试模块的路径, 存放在 testCase 包中; 第二个参数 pattern 用来获取 testCase 包中所有以 test 开头的模块文件, 会获取到 test_baidu.py 和 test_sina.py; 第三个参数 top_level_dir 在调用的时候直接给默认值 None。discover 方法的代码如下:

```
def discover(self, start_dir, pattern='test*.py', top_level_dir=None):
```

运行以上 allTests.py 文件后, 测试结果如图 4-8-1 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter4/allTests.py
test_baidu_title (test_baidu.BaiduTest)
验证: 测试百度首页的title是否正确 ... ok
test_username_password_null (test_sina.SinaTest)
验证: 新浪登录页面用户名和密码为空错误提示信息 ... ok

-----

Ran 2 tests in 20.008s
|
OK

Process finished with exit code 0
```

图 4-8-1

4.9 生成测试报告

运行 `allTests.py` 文件后得到的测试结果不够专业，无法直接提交，因此需要借助第三方库生成 HTML 格式的测试报告。这里用到的库是 `HTMLTestRunner.py`，下载地址是：

<https://github.com/tungwaiyip/HTMLTestRunner>

下载 `HTMLTestRunner.py` 文件后，把该文件放到 Python 安装路径的 Lib 子文件夹中，例如，`C:\Python36-32\Lib` 目录下。创建 `report` 文件夹，与 `testCase` 包放在同一个目录下，继续完善 `allTests.py` 文件，最终生成测试报告，最终的 `allTests.py` 代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import os
import HTMLTestRunner
import time

def allTests():
    ''' 获取所有要执行的测试用例 '''
    suite=unittest.defaultTestLoader.discover(
        start_dir=os.path.join(os.path.dirname(__file__),'testCase'),
        pattern='test_*.py',
        top_level_dir=None
    )
    return suite

def getNowTime():
    ''' 获取当前的时间 '''
    return time.strftime('%Y-%m-%d %H_%M_%S',time.localtime(time.time()))

def run():
    fileName=os.path.join(os.path.dirname(__file__),'report',
        getNowTime()+'.report.html')
```

```

fp=open(fileName, 'wb')
runner=HTMLTestRunner.HTMLTestRunner(
    stream=fp,
    title='UI 自动化测试报告',
    description='UI 自动化测试报告详细信息')
runner.run(allTests())

if __name__ == '__main__':
    run()

```

注解：在以上完善后的 `allTests.py` 文件中其中导入了 `sys`、`HTMLTestRunner` 库。`getNowTime` 方法用来获取当前时间，每一次生成的测试报告如果文件名称一致，由于加上了最新时间信息，便可以根据文件名称确认哪个是最新的测试报告；`run` 方法用来执行测试套件中的测试用例和生成测试报告。针对 `HTMLTestRunner.py` 文件，在 `Python3` 中需要对代码进行修改，否则，执行 `allTests.py` 后就会报错。

再次运行 `allTests.py` 文件后，在 `report` 文件夹下生成了最新的测试报告，`report` 的目录如图 4-9-1 所示。

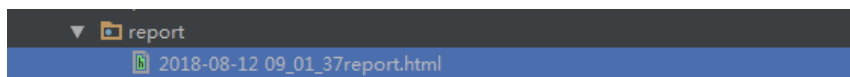


图 4-9-1

打开该 `HTML` 文件，可以看到基于 `HTML` 的测试报告，如图 4-9-2 所示。

UI自动化测试报告

Start Time: 2018-08-12 09:01:37

Duration: 0:00:21.631432

Status: Pass 2

UI自动化测试报告详细信息

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
test_baidu.BaiduTest	1	1	0	0	Detail
test_sina.SinaTest	1	1	0	0	Detail
Total	2	2	0	0	

图 4-9-2

注解：在图 4-9-2 所示的基于 HTML 的测试报告中，我们可以看到开始时间、执行时间、测试用例通过数、失败数、错误数，点击 **Detail** 还可以看到更详细的信息，如图 4-9-3 所示。

UI自动化测试报告

Start Time: 2018-08-12 09:01:37

Duration: 0:00:21.631432

Status: Pass 2

UI自动化测试报告详细信息

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
test_baidu.BaiduTest	1	1	0	0	Detail
test_baidu_title: 验证：测试百度首页的title是否正确		pass			
test_sina.SinaTest	1	1	0	0	Detail
test_username_password_null: 验证：新浪登录页面用户名和密码为空错误提示信息		pass			
Total	2	2	0	0	

图 4-9-3

4.10 代码覆盖率统计实战

Coverage.py 是 Python 程序代码覆盖率的测试工具，用于监视程序执行了哪些代码，未执行哪些代码。在 Python3 中，首先需要通过 `pip3 install coverage` 来安装它，安装过程如图 4-10-1 所示。

```

C:\Users\Administrator>pip3 install coverage
Collecting coverage
  Downloading https://files.pythonhosted.org/packages/5e/7c/b6441f0fd4070f4be7c23aaac4808932ed38aa7f076105e780355e85657f/coverage-4.5.1-cp36-cp36m-win32.whl (180kB)
    45% |#####| 81kB 12kB/s eta 0:00:00
    51% |#####| 92kB 11kB/s eta 0:00:00
    56% |#####| 102kB 11kB/s eta 0:00:00
    62% |#####| 112kB 8.4kB/s eta 0:00:00
    68% |#####| 122kB 6.1kB/s eta 0:00:00
    73% |#####| 133kB 4.8kB/s eta 0:00:00
    79% |#####| 143kB 3.9kB/s eta 0:00:00
    85% |#####| 153kB 3.8kB/s eta 0:00:00
    91% |#####| 163kB 3.8kB/s eta 0:00:00
    96% |#####| 174kB 3.8kB/s eta 0:00:00
   100% |#####| 184kB 3.8kB/s eta 0:00:00
Installing collected packages: coverage
Successfully installed coverage-4.5.1
C:\Users\Administrator>

```

图 4-10-1

安装 coverage 后运行 allTests.py 文件，程序会运行所有以 test 开头的测试模块的文件。到 allTests.py 模块的路径下运行以下代码：

```
coverage3 run allTests.py
```

再次执行 coverage html，也就是代码覆盖率统计，通过 html 的文件查看，执行的命令如图 4-10-2 所示。

```
D:\git\Python\ActualCombat\Chapter4>coverage html  
D:\git\Python\ActualCombat\Chapter4>
```

图 4-10-2

执行后，在 Chapter4 目录下会生成一个 htmlcov 文件夹，在该文件夹里面显示的是代码覆盖率统计的文件，如图 4-10-3 所示。

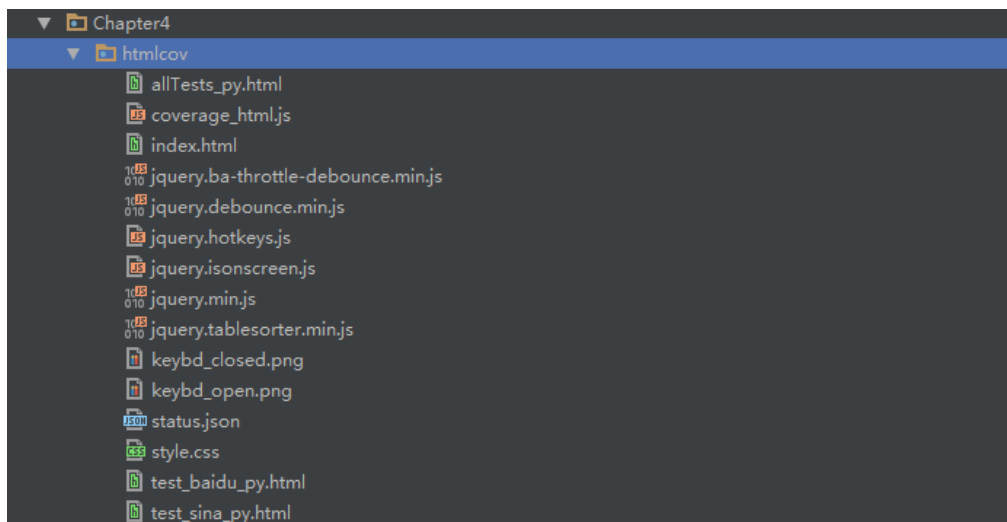


图 4-10-3

点击打开 index.html 文件，显示的是每个文件运行代码的覆盖率统计，如图 4-10-4 所示。

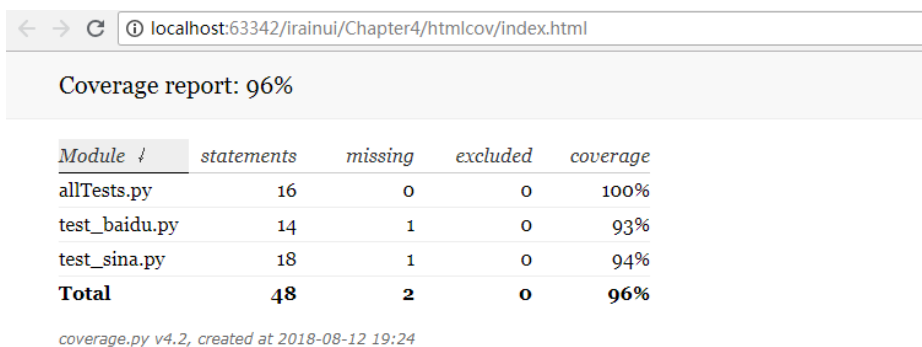


图 4-10-4

点击任意一个模块文件，就会显示该模块执行的代码，如点击 test_sina.py 文件，代码如图 4-10-5 所示。

Coverage for **test_sina.py** : 94%

18 statements 17 run 1 missing 0 excluded

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuyya

import unittest
from selenium import webdriver

class SinaTest(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def test_username_password_null(self):
        '''验证：新浪登录页面用户名和密码为空错误提示信息'''
        self.driver.find_element_by_id('freename').send_keys('')
        self.driver.find_element_by_id('freepassword').send_keys('')
        self.driver.find_element_by_link_text(u'登录').click()
        divError=self.driver.find_element_by_xpath('/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]').text
        self.assertEqual(divError,u'请输入邮箱名')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

图 4-10-5

第 5 章 Jenkins 实战

5.1 Jenkins 简述及部署

1. Jenkins 简述

Jenkins 是一款自包含的开源自动化服务器，可用于自动执行与构建、测试、交付和部署软件有关的各种任务。Jenkins 可以通过本机系统软件包 Docker 进行安装，还可以在任何安装了 Java 运行环境（JRE）的计算机上独立运行。

2. Jenkins 部署

部署 Jenkins 首先需要搭建 Java 环境，搭建 Java 环境后，然后下载 tomcat，从 <https://jenkins.io/index.html> 链接页面上下载 jenkins.war，下载成功后把 jenkins.war 文件放在 tomcat 的 webapps 目录下，启动 tomcat。在浏览器中访问 <http://localhost:8080/jenkins>，首次访问会提示安装插件，插件安装成功后，点击“完成”按钮。登录需要密码，密码地址在 C:\Users\Administrator\jenkins\secrets 目录下的 initialAdminPassword 中，打开文件后，将密码复制到 Jenkins 的密码输入框中，点击“登录”按钮，成功登录到 Jenkins 中。

5.2 Jenkins 实战

1. Job 实战

下面结合前面的实例代码，把代码部署在 Jenkins 中。首先创建 Job，在 Jenkins 首页点击“新建任务”按钮，将任务名称填写为 Book，选择“构建一个自由风格的软件项目”后点击“确定”按钮。创建任务后的界面如图 5-2-1 所示。

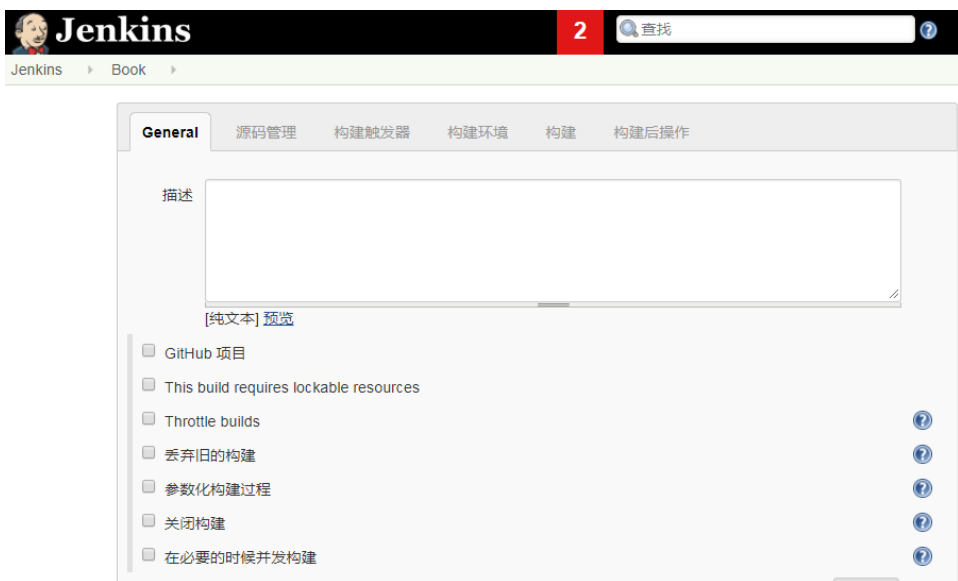


图 5-2-1

在“构建”选项中点击“增加构建步骤”的下拉框选项，选择“执行 Windows 批处理命令”后，如图 5-2-2 所示。



图 5-2-2

在命令输入框中输入要执行的文件 allTests.py 的路径，如图 5-2-3 所示。



图 5-2-3

点击“保存”按钮后，保存后 Book 任务的页面如图 5-2-4 所示。



图 5-2-4

在图 5-2-4 中选择“立即构建”选项，执行成功后，控制台的输出结果如图 5-2-5 所示。

控制台输出

```
由用户 无涯 启动
构建中 在工作空间 C:\Users\Administrator\.jenkins\workspace\Book 中
[Book] $ cmd /c call E:\tools\apache-tomcat-8.0.33\temp\jenkins4396414030313671534.bat

C:\Users\Administrator\.jenkins\workspace\Book>cd D:\git\github\book

C:\Users\Administrator\.jenkins\workspace\Book>d:

D:\git\github\book>python allTests.py
..<_io.TextIOWrapper name='<stderr>' mode='w' encoding='cp936'>
Time Elapsed: 0:00:29.256673

D:\git\github\book>exit 0
Finished: SUCCESS
```

图 5-2-5

2. HTML Report 实战

图 5-2-5 中的执行结果是 SUCCESS，同时生成的 HTML 的测试报告存放在 report 文件夹目录下，如果可以在 Jenkins 的平台中直接查看测试报告，那么就不需要执行完成后，再打开 HTML 去看。在 Book 任务的配置页面中，在构建后操作点击“增加构建后操作步骤”的下来框，在下拉框选项中选择“Publish HTML reports”，如图 5-2-6 所示。

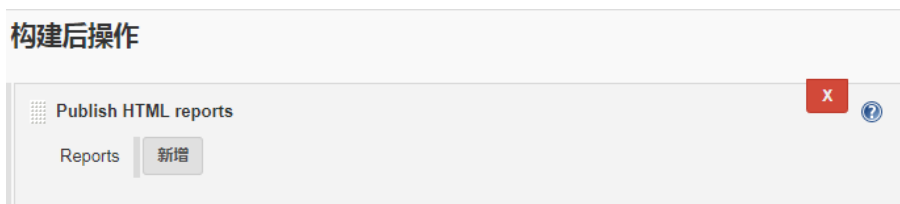


图 5-2-6

在 Publish HTML reports 中点击“增加”按钮，在 Reports 中的 HTML directory tory to archive 填写测试报告的目录，在 Index page[s]中添加获取所有的 HTML 测试报告信息，填写后如图 5-2-7 所示。

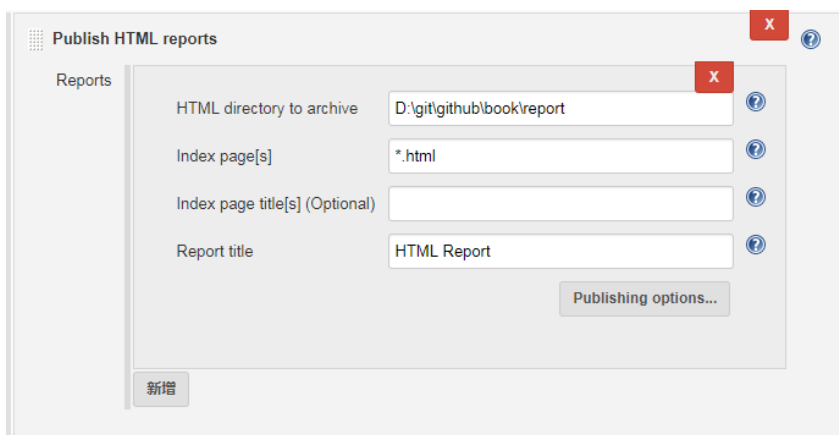


图 5-2-7

注解：在 Index page[s]输入框中填写的是获取所有 HTML 的测试报告，所以填写 *.html。

点击“保存”按钮后，在图 5-2-4 中再次选择“立即构建”选项，执行后的 Book 任务页面主页如图 5-2-8 所示。



图 5-2-8

在图 5-2-8 中可以看到，在 Book 的详情中新增了 HTML Report。点击 HTML Report，跳转到所有测试报告页面，如图 5-2-9 所示。

Book html2

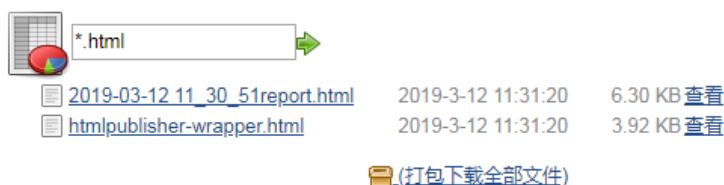


图 5-2-9

在图 5-2-9 中点击 HTML 的测试报告文件，测试结果如图 5-2-10 所示。

UI自动化测试报告

Start Time: 2019-03-12 11:30:51

Duration: 0:00:28.580635

Status: Pass 2

UI自动化测试报告详细信息

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
testCase.test_baidu.BaiduTest	1	1	0	0	Detail
test_baidu_title: 验证：测试百度首页的title是否正确					pass
testCase.test_sina.SinaTest	1	1	0	0	Detail
test_username_password_null: 验证：新浪登录页面用户名和密码为空错误提示信息					pass
Total	2	2	0	0	

图 5-2-10

3. Allure 的实战

前面介绍了基于 Publish html reports 生成测试报告，但这份报告看起来并不是很直观，而使用 Allure 插件结合 Jenkins、Pytest 测试框架，可生成简洁美观的测试报告。Allure 的 Web 报表表单不仅以简明的形式呈现，而且允许开发人员在开发过程中提取到最大的、可用的测试信息。

使用 Allure 首先需要在 Jenkins 的“插件管理”中安装插件 Allure 插件，Allure 安装成功后，可以在管理插件的“已安装”菜单栏中看到，如图 5-2-11 所示。



图 5-2-11

接下来需要在系统管理的“全局工具配置”中安装 JDK 和 Allure Commandline，在 JDK 安装中，选择“新增 JDK”选项，取消选择“自动安装”选项后，填写 JDK 的别名和 JAVA_HOME 的路径，如图 5-2-12 所示。



图 5-2-12

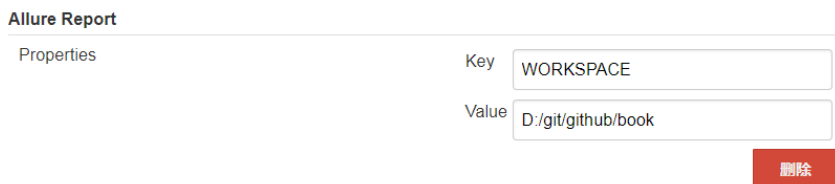
在 Allure Commandline 页面中选择“Allure Commandline 安装”选项，选择自动安装，在“版本”中选择要安装的版本，这里选择“2.7.0”，如图 5-2-13 所示。



图 5-2-13

接下来在“系统管理”的“系统设置”选项中配置 Allure Report 目录，在

“系统设置”中找到“Allure Report”选项，点击“新增”按钮，在 Key 中填写 WORKSPACE，在 Value 中填写要执行项目的路径，如图 5-2-14 所示。



Allure Report

Properties

Key WORKSPACE

Value D:/git/github/book

删除

图 5-2-14

注解：在 Key 中填写 WORKSPACE，在 Value 中指定测试报告的路径。

接下来在 Windows 打开 dos 窗口安装 Pytest 所需的第三方库，安装的命令为：

```
pip install pytest
pip install pytest-allure-adaptor
```

安装命令执行后如图 5-2-15 所示。

```
C:\Users\Administrator>pip install pytest-allure-adaptor
Collecting pytest-allure-adaptor
  Using cached https://files.pythonhosted.org/packages/2e/94/862ca2f86f3644fd6687e254518ff57fe729676172ef37594913e88a2e3c/pytest_allure_adaptor-1.7.10-py3-none-any.whl
Requirement already satisfied: namedlist in c:\python36-32\lib\site-packages (from pytest-allure-adaptor) (1.7)
Requirement already satisfied: pytest>=2.7.3 in c:\python36-32\lib\site-packages (from pytest-allure-adaptor) (4.3.0)
Requirement already satisfied: lxml>=3.2.0 in c:\python36-32\lib\site-packages (from pytest-allure-adaptor) (4.2.1)
Requirement already satisfied: six>=1.9.0 in c:\python36-32\lib\site-packages (from pytest-allure-adaptor) (1.10.0)
Requirement already satisfied: enum34 in c:\python36-32\lib\site-packages (from pytest-allure-adaptor) (1.1.6)
Requirement already satisfied: attrs>=17.4.0 in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (17.4.0)
Requirement already satisfied: pluggy>=0.7 in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (0.8.1)
Requirement already satisfied: more-itertools>=4.0.0; python_version > "2.7" in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (4.2.0)
Requirement already satisfied: setuptools in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (28.8.0)
Requirement already satisfied: atomicwrites>=1.0 in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (1.1.5)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (0.3.9)
Requirement already satisfied: py>=1.5.0 in c:\python36-32\lib\site-packages (from pytest>=2.7.3->pytest-allure-adaptor) (1.5.4)
Installing collected packages: pytest-allure-adaptor
Successfully installed pytest-allure-adaptor-1.7.10
```

图 5-2-15

配置和安装相关的插件后，再次配置 Book 任务。在“执行 Windows 批处理命令”的输入框修改使用 Pytest 用以执行，修改后的构建如图 5-2-16 所示。



图 5-2-16

在“增加构建后操作步骤”下拉框选项中选择“Allure Report”选项，点击“新增”按钮，在 Path 文本框中填写 report（Path 中填写的 report 与项目中 report 测试报告目录名称必须一致），如图 5-2-17 所示。

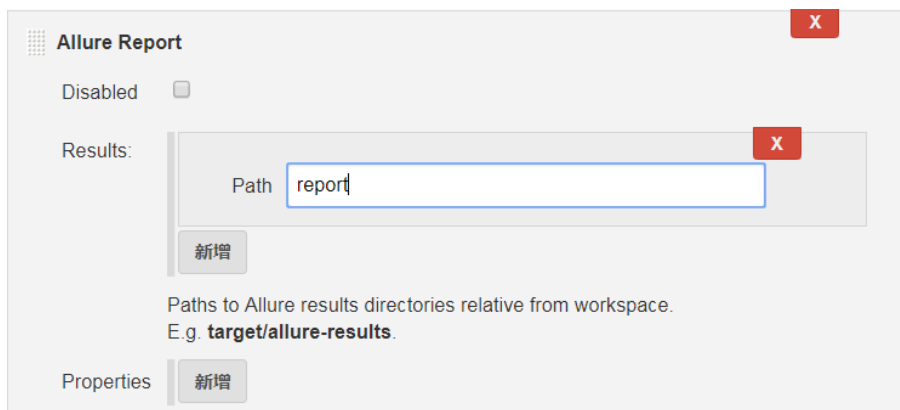


图 5-2-17

注解：Path 中的 report 与系统管理下系统设置中 Value 的 report 路径名称务必一致。

点击“保存”按钮后，在图 5-2-8 中选择“立即构建”选项，执行成功后输出的结果信息如图 5-2-18 所示。

```

D:\git\github\book\testCase>python -m pytest --alluredir $(WORKSPACE)/report
===== test session starts =====
platform win32 -- Python 3.7.2, pytest-4.0.0, py-1.8.0, pluggy-0.9.0
rootdir: D:\git\github\book\testCase, inifile:
plugins: cov-2.6.1, allure-adaptor-1.7.10
collected 2 items

test_baidu.py . [ 50%]
test_sina.py . [100%]

===== 2 passed in 52.06 seconds =====

D:\git\github\book\testCase>exit 0
[Book] $
C:\Users\Administrator\.jenkins\tools\ru.yandex.qatools.allure.jenkins.tools.AllureCommandlineInstallation\Allure\bin\allure.bat generate C:\Users\Administrator\.jenkins\workspace\Book\report -c -o
C:\Users\Administrator\.jenkins\workspace\Book\allure-report
Report successfully generated to C:\Users\Administrator\.jenkins\workspace\Book\allure-report
Allure report was successfully generated.
Creating artifact for the build.
Artifact was added to the build.
[htmlpublisher] Archiving HTML reports...
[htmlpublisher] Archiving at PROJECT level D:\git\github\book\report to
C:\Users\Administrator\.jenkins\jobs\Book\htmlreports\HTML_20Report
Finished: SUCCESS

```

图 5-2-18

注解：在输出的信息中，可以看到执行的测试用例数，测试模块，以及执行完成后生成的测试报告。

返回到任务 Book 的详情页，可以看到已经有了 Allure Report 的测试报告，如图 5-2-19 所示。



图 5-2-19

点击图 5-2-19 中的 “Allure Report” 后，可以查看最新生成的测试报告，如图 5-2-20 所示。

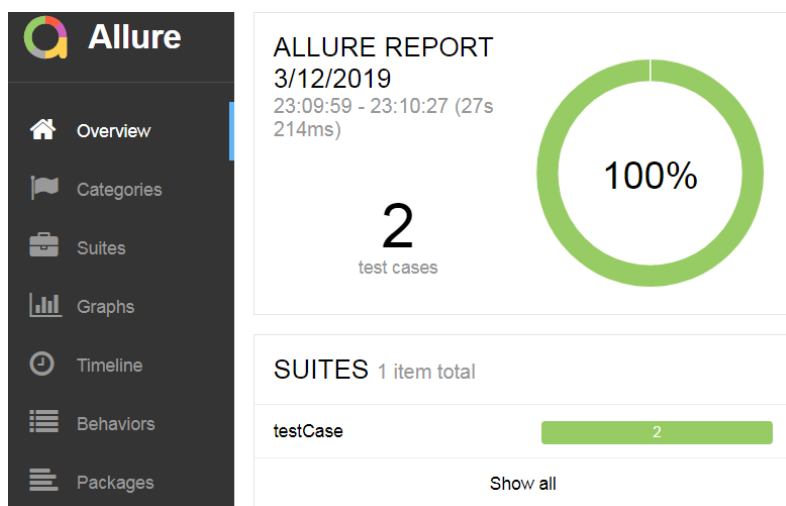


图 5-2-20

点击图 5-2-20 中的 “Graphs” 部分，如图 5-2-21 所示。

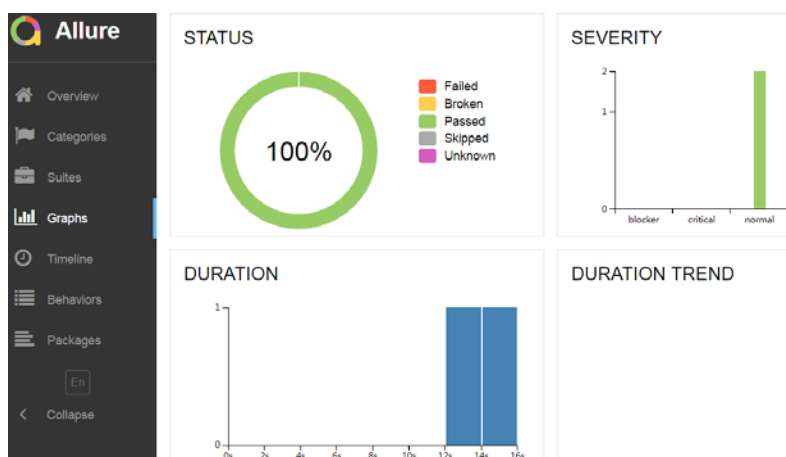


图 5-2-21

在图 5-2-21 中我们可以看到，测试报告不仅更加直观而且看起来比较漂亮。

4. 定时任务

Jenkins 也可根据工作需求设定执行任务的时间，例如，每两分钟执行某个

Job 或者每隔一小时执行某个 Job。这里创建一个 Job 名为 Crontab，在“构建触发器”中选择“定时构建”，在里面填写*/2 * * * *，如图 5-2-22 所示。



图 5-2-22

点击“保存”按钮后，每隔两分钟就会执行一次，如图 5-2-23 所示。



图 5-2-23

5. 安全域的配置

某些时候，可能你只允许某些同事查看任务展示，而不允许他们拥有构建、删除、编辑等权限，这时可使用到安全域的配置。首先在“全局安全配置”的“安全域”中选择“允许用户注册”选项，点击“保存”按钮后，退出 admin 账

户。注册用户 weke，注册成功后，再次使用 admin 登录系统，只授予 weke 用户查看任务和读任务的权限，不给授予执行的权限。在“系统管理”的“全局安全配置”中，在“授权策略”页面中选择“安全矩阵”选项，然后点击“Add user or group”，填写添加的账户 weke，只选择“Read”权限，如图 5-2-24 所示。



图 5-2-24

点击“保存”按钮后，weke 账户授权策略配置成功后，退出 admin 用户，用 weke 用户登录后，则该用户只允许查看任务的视图，无构建等权限，如图 5-2-25 所示。



图 5-2-25

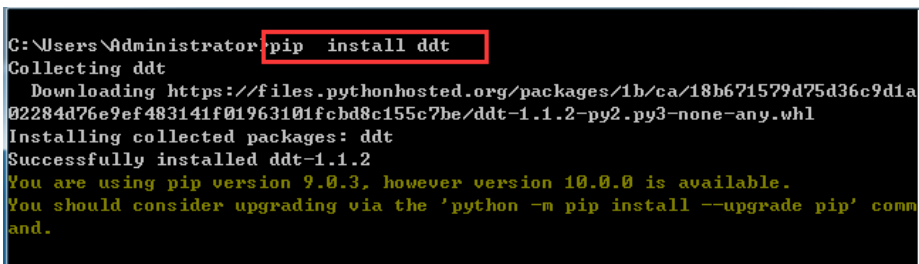
注解：在图 5-2-25 中，可以看到该用户没有配置和立即构建的入口，只拥有查看视图和查看测试结果的权限。

第 6 章 数据驱动

在 UI 自动化测试中，自动化测试的数据如何分离和高效的维护，页面元素又怎样分离和维护，都是自动化测试人员需要面对的难点问题，本章节的重点就是解决这些问题。

6.1 ddt 实战

ddt 是 Python 的第三方库。ddt 模块提供了创建数据驱动测试，关于该模块详细的信息建议到官方查看，地址为：<https://pypi.org/project/ddt/>，安装的命令是：pip install ddt，如图 6-1-1 所示。



```
C:\Users\Administrator>pip install ddt
Collecting ddt
  Downloading https://files.pythonhosted.org/packages/1b/ca/18b671579d75d36c9d1a02284d76e9ef483141f01963101fcbd8c155c7be/ddt-1.1.2-py2.py3-none-any.whl
Installing collected packages: ddt
Successfully installed ddt-1.1.2
You are using pip version 9.0.3, however version 10.0.0 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

图 6-1-1

ddt 安装成功后，在 Python 的命令行环境中即可导入。在 ddt 模块中，@data 表示元组的列表数据，@unpack 表示用来解压元组到多个参数。ddt 库应用在 UI 自动化测试中，实现编写一条测试用例的代码验证多个测试点。例如，在新浪登录页面中，存在多种测试需求，如用户名和密码输入框都为空，用户名为空、密码不为空，密码为空、用户名不为空，分别会返回不同的错误提示信息。下面通过 ddt 来实现以下实例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
```

```

import unittest
from selenium import webdriver
from ddt import data, unpack, ddt

@ddt
class SinaLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    @data((' ', ' ', u'请输入邮箱名'), (' ', 'admin', u'请输入邮箱名'), ('admin', ' ', u'您输入的邮箱名格式不正确'))
    @unpack
    def test_login(self, username, password, result):
        '''验证：测试新浪邮箱登录N种情况'''
        self.driver.find_element_by_id('freename').send_keys(username)
        self.driver.find_element_by_id('freepassword').
send_keys(password)
        self.driver.find_element_by_link_text(u'登录').click()
        divText=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
').text
        self.assertEqual(divText, result)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：以上代码主要反映了 ddt 在测试用例中的应用，在@data 中数据类型是元组，可以看到不同情况下的测试数据，也就是以下三个测试点：

- (1) 用户名和密码为空，点击“登录”按钮，验证错误提示信息是否是“请输入邮箱名”；
- (2) 用户名为空，密码不为空，点击“登录”按钮，验证错误是否是“请输入邮箱名”；
- (3) 输入错误的用户名，点击“登录”按钮，验证错误是否是“您输入的邮箱格式不正确”。

在测试用例中，test_login 中有三个参数，分别是 username，password，result，分别与@data 的元组数据一一对应。

与执行后的结果如图 6-1-2 所示。

```
test_login_1_____u_u8bf7_u8f93_u5165_u90ae_u7bb1_u540d__ (_main__ SinaLogin)
验证: 测试新浪邮箱登录II中情况 ... ok
test_login_2_____admin_u_u8bf7_u8f93_u5165_u90ae_u7bb1_u540d__ (_main__ SinaLogin)
验证: 测试新浪邮箱登录II中情况 ... ok
test_login_3__admin_____u_u60a8_u8f93_u5165_u7684_u90ae_u7bb1_u540d_u683c_u5f0f_u4e0d_u6b63_u786e__ (_main__ SinaLogin)
验证: 测试新浪邮箱登录II中情况 ... ok

-----

Ran 3 tests in 24.853s

OK
```

图 6-1-2

在图 6-1-2 中可以看到，三个测试用例都被执行并且通过。这就是 ddt 的优秀之处，而我们写的测试代码只是一个测试用例的代码。

以上实例是把数据放在@data 中，可以把@data 中的数据分离到一个方法中，例如，存储在列表里。下面实现把@data 中的数据放在一个列表中，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver
from ddt import data,unpack,ddt

def getData():
    ''' 数据分离出来放到列表中'''
    return [
        ['', '',u'请输入邮箱名'],
        ['', 'admin',u'请输入邮箱名'],
        ['admin','',u'您输入的邮箱名格式不正确']]

@ddt
class SinaLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)
```

```

def tearDown(self):
    self.driver.quit()

@data(*getData())
@unpack
def test_login(self, username, password, result):
    '''验证：测试新浪邮箱登录N种情况'''
    self.driver.find_element_by_id('freename').send_keys(username)
    self.driver.find_element_by_id('freepassword').
send_keys(password)
    self.driver.find_element_by_link_text(u'登录').click()
    divText=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
').text
    self.assertEqual(divText,result)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在以上代码中可以看到@data 中的数据被分离到了 getData 函数中，在@data 中调用函数 getData 时增加了“*”，是因为@data 要求的数据类型是元组，加“*”后便把 getData 函数返回的数据列表类型变为元组类型。

6.2 Txt 实战

在 UI 自动化测试中，也可以把测试的数据存储在后缀为 txt 的记事本文件中，对记事本的操作一般包括读取文件和写入文件。其中，r 为只读属性，r+是读写属性，w 是只写属性，w+是读写属性，a+是读写方式。下面通过一个实例，实现向记事本中写入文件内容再读取出来，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

def txt():
    '''对记事本的操作'''
    f=open('file.txt','w')
    f.write('无涯您好!')
    f.close()
    f=open('file.txt','r')
    print f.read()
    f.close()

```

注解：`f=open('file.txt','w')`表示不管当前路径下是否存在 `file.txt`，都会创建一个记事本文件 `file.txt`，文件打开后切记要关闭。函数 `txt` 执行后首先在 `file.txt` 文件中写入“无涯您好！”内容，读取后也是该文件内容。

不管是写入文件还是读取文件内容都需要按顺序进行打开文件和关闭文件的操作，有时会忘记编写关闭文件的步骤。为解决这一问题，可通过 `with` 的方式实现系统内部的文件关闭处理而不需要专门关闭文件，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

def txt():
    '''对记事本的操作'''
    with open('file.txt','w') as f:
        f.write('无涯您好!')
    with open('file.txt','r') as f:
        print f.read()
```

下面实现把测试过程中用到的数据存储在 `txt` 的记事本中。还是以新浪登录为例，记事本 `sina.txt` 文件中的内容如下：

```
admin
请输入邮箱名
您输入的邮箱名格式不正确
```

`sina.txt` 文件就在当前目录下，下面编写记事本文件在新浪登录自动化测试中的应用，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import unittest
from selenium import webdriver
import os

def getTxtData(index):
    with open(os.path.join(os.path.dirname(__file__),'sina.txt'),'r',encoding='utf-8') as f:
        d=f.readlines()
    return d[index]
```

```

class SinaLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def login(self, username, password):
        self.driver.find_element_by_id('freename').send_keys(username)
        self.driver.find_element_by_id('freepassword').
send_keys(password)
        self.driver.find_element_by_link_text(u'登录').click()

    def divText(self):
        divText=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
')
        return (divText.text).encode('utf-8')

    def test_username_password_null(self):
        '''验证: 测试用户名和密码都为空的错误提示信息'''
        self.login(getTxtData(0),getTxtData(0))
        self.assertTrue(self.divText(),getTxtData(2))

    def test_sina_password_null(self):
        '''验证: 测试用户名为空密码不为空的错误提示信息'''
        self.login(getTxtData(0),getTxtData(1))
        self.assertTrue(self.divText(),getTxtData(2))

    def test_sina_username_format(self):
        '''验证: 测试用户名邮箱格式不正确的错误提示信息'''
        self.login(getTxtData(1),getTxtData(1))
        self.assertEqual(self.divText(),getTxtData(3))

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在代码中，读取记事本文件的函数是 `getTxtData`，该函数的形式参数是 `index`，也就是文件按行读取后，想要哪行数据，在调用该函数的时候，直接传对应的行数的索引即可。把登录单独写成 `login` 方法，把返回的错误提示信

息放在 `divText` 方法中，这样，在三个测试用例中只需要写很少的代码，直接调用 `login` 方法和 `divText` 方法即可。

6.3 Csv 实战

Python 提供了对 csv 文件处理的模块，csv 文件全称为 Comma-Separated Values，csv 是通用的、相对简单的文件格式，其文件以纯文件形式存储数据。

1. csv 文件读取处理

在 `testCase` 包中创建 `test.csv` 文件，需要注意：创建 csv 为后缀的文件时，一定要先创建 `test.xlsx`，打开文件后，再另存为 `test.csv`，如图 6-3-1 所示。

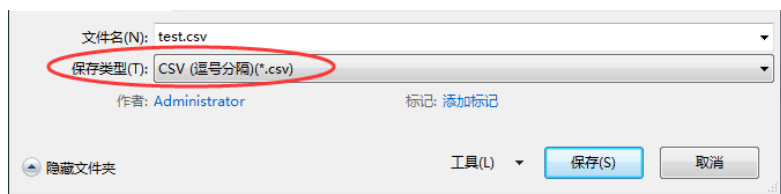


图 6-3-1

如果直接把 `test.xlsx` 修改为 `test.csv`，打开 csv 的文件就会出现错误提示：`_csv.Error: line contains NULL byte`。在 `test.csv` 中存储的文件内容，如图 6-3-2 所示。

用户名	密码	错误提示信息
		请输入邮箱名
	admin	请输入邮箱名
admin		您输入的邮箱名格式不正确

图 6-3-2

下面来实现把以上 csv 文件里的内容读取出来，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

import csv
import os

def readCsv(row, col):
```

```

rows = []
with open(os.path.join(os.path.dirname(__file__), 'test.csv')) as f:
    reader = csv.reader(f)
    next(reader, None)
    for iter in reader:
        rows.append(iter)
return ''.join(rows[row][col]).decode('gb2312')

```

注解：在以上代码中，首先把读取的文件内容放在 `rows` 的列表中，`".join(rows[row][col])"`是把列表 `row` 转换为字符串，`decode('gb2312')`则把 `unicode` 编码的字符串转换为 `gb2312`，这样存储的中文读取出来后结果显示中文，类型是 `unicode`。

2. csv 在自动化中的实战

还是以新浪登录为例，实现把测试中的数据存储到 `csv` 文件中，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

import csv
import unittest
from selenium import webdriver

def readCsv(row,col):
    rows = []
    with open('test.csv') as f:
        reader = csv.reader(f)
        next(reader, None)
        for iter in reader:
            rows.append(iter)
    return ''.join(rows[row][col]).decode('gb2312')

class SinaLogin(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):

```

```

self.driver.quit()

def login(self, username, password):
    self.driver.find_element_by_id('freename').send_keys(username)
    self.driver.find_element_by_id('freepassword').
send_keys(password)
    self.driver.find_element_by_link_text(u'登录').click()

def divText(self):
    divText=self.driver.find_element_by_xpath(
'/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]')

    return divText.text

def test_username_password_null(self):
    '''验证: 测试用户名和密码都为空的错误提示信息'''
    self.login(readCsv(0,0),readCsv(0,1))
    self.assertEqual(self.divText(),readCsv(0,2))

def test_sina_password_null(self):
    '''验证: 测试用户名为空密码不为空的错误提示信息'''
    self.login(readCsv(1,0),readCsv(1,1))
    self.assertTrue(self.divText(),readCsv(1,2))

def test_sina_username_format(self):
    '''验证: 测试用户名邮箱格式不正确的错误提示信息'''
    self.login(readCsv(2,0),readCsv(2,1))
    self.assertEqual(self.divText(),readCsv(2,2))

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

代码执行后的结果如图 6-3-3 所示。

```

test_sina_password_null (__main__.SinaLogin)
验证: 测试用户名为空密码不为空的错误提示信息 .... ok
test_sina_username_format (__main__.SinaLogin)
验证: 测试用户名邮箱格式不正确的错误提示信息 .... ok
test_username_password_null (__main__.SinaLogin)
验证: 测试用户名和密码都为空的错误提示信息 .... ok

-----
Ran 3 tests in 26.390s

OK

```

图 6-3-3

6.4 Excel 实战

在 UI 自动化测试中，处理 Excel 文件需要使用到第三方库 xlrd，xlrd 需要单独安装，安装的命令是：

```
pip install xlrd
```

安装界面如图 6-4-1 所示。

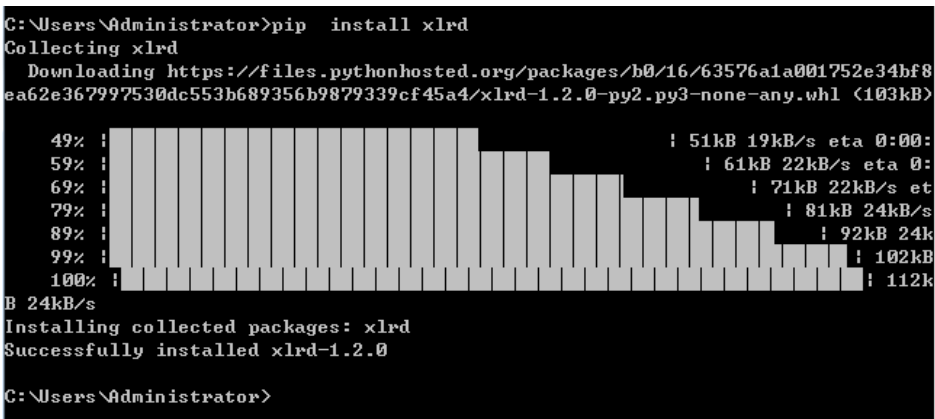


图 6-4-1

1. excel 文件读取处理

接下来介绍如何读取 Excel 中的数据。创建一个 Excel 文件 sina.xlsx，把新浪邮箱登录为空的错误信息储存在 Excel 文件中，见存储的数据如图 6-4-2 所示。

用户名	密码	错误提示信息
WuYa	admin	请输入邮箱名
		您输入的邮箱名格式不正确

图 6-4-2

实现读取 excel 中数据的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8 -*-

#author:wuya
```



```
import xlrd

def readExcel(row):
    '''
    :param row: 该参数表示行
    '''
    book=xlrd.open_workbook('sing.xlsx','r')
    table=book.sheet_by_index(0)
    return table.row_values(row)
```

注解：以上代码中，第一行代码是打开 sing.xlsx 文件；第二行是具体到 excel 文件的 sheet，这里以索引的方式处理，因为数据是在第一个 sheet 的，所以索引为 0；最后一行 row_values()是读取 XX 行的数据，上传的参数应该是具体的行。调用以上函数后，返回的数据类型是列表，调用后返回的结果如图 6-4-3 所示。

```
print('执行后数据为:{0},数据类型为:{1}'.format(readExcel(1),type(readExcel(1))))
excelTest
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter5/excelTest.py
执行后数据为:[' ',' ','请输入邮箱名'],数据类型为:<class 'list'>

Process finished with exit code 0
|
```

图 6-4-3

2. Excel 在自动化中的实战

下面以实现新浪邮箱登录为测试用例，两个测试点分别是用户名为空和登录邮箱格式错误，把测试的数据存储在 Excel 中（具体见以上 Excel 的数据），测试用到的数据直接从 Excel 中读取，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import xlrd
import unittest
from selenium import webdriver
```

```

import time as t

def readExcel(row):
    '''
    :param row: 该参数表示行
    '''
    book=xlrd.open_workbook('sina.xlsx','r')
    table=book.sheet_by_index(0)
    return table.row_values(row)

class SinaTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def login(self,username,password):
        t.sleep(2)
        #邮箱账号输入框
        self.driver.find_element_by_id('freename').send_keys(username)
        t.sleep(2)
        self.driver.find_element_by_id('freepassword').send_keys(password)
        t.sleep(2)
        self.driver.find_element_by_link_text('登录').click()

    def getLoginError(self):
        '''返回点击“登录”按钮后的错误提示信息'''
        t.sleep(2)
        loginError=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
')
        return loginError.text

    def test_sina_login_null(self):
        '''登录业务: 用户名和密码为空的错误提示信息'''
        self.login(readExcel(1)[0],readExcel(1)[1])
        self.assertEqual(self.getLoginError(),readExcel(1)[2])

    def test_sina_login_fromat(self):
        '''登录业务: 用户名登录格式错误提示信息'''

```

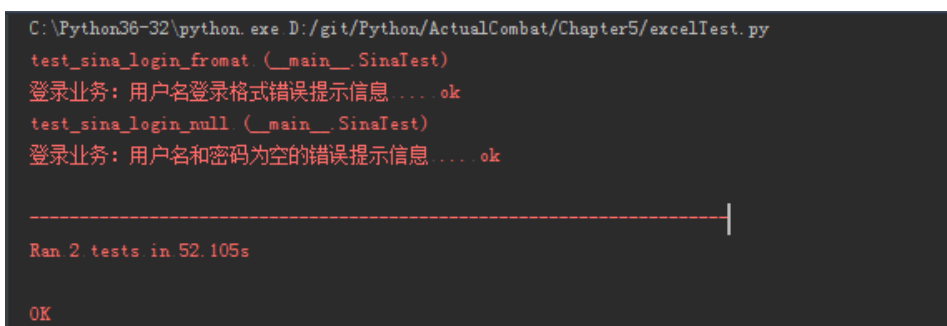
```

        self.login(readExcel(2)[0],readExcel(2)[1])
        self.assertEqual(self.getLoginError(),readExcel(2)[2])

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在以上代码中，把登录和获取错误的提示信息单独写了个方法，并且直接调用，这样减少了多余代码的重复写入，代码执行后的结果如图 6-4-4 所示。



```

C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter5/excelTest.py
test_sina_login_fromat (__main__.SinaTest)
登录业务：用户名登录格式错误提示信息 ... ok
test_sina_login_null (__main__.SinaTest)
登录业务：用户名和密码为空的错误提示信息 .....ok

-----|
Ran 2 tests in 52.105s

OK

```

图 6-4-4

3. ddt 在 Excel 中的实战

前面详细地介绍了 ddt 库在自动化测试中的应用，这里，将 ddt 和 Excel 进行整合，让测试代码更加简单，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import xlrd
import unittest
from selenium import webdriver
import time as t
from ddt import ddt,data,unpack

def readExcel(row):
    '''
    :param row: 该参数表示行
    '''
    book=xlrd.open_workbook('sina.xlsx','r')
    table=book.sheet_by_index(0)

```

```

        return table.row_values(row)

def readExcels():
    ''' 读取 excel 数据添加到 rows 列表中'''
    rows=[]
    book=xlrd.open_workbook('sina.xlsx','r')
    sheet=book.sheet_by_index(0)
    for row in range(1,sheet.nrows):
        rows.append(sheet.row_values(row,0,sheet.ncols))
    return rows

@ddt
class SinaTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get('http://mail.sina.com.cn/')
        self.driver.implicitly_wait(30)

    def tearDown(self):
        self.driver.quit()

    def login(self,username,password):
        t.sleep(2)
        #邮箱账号输入框
        self.driver.find_element_by_id('freename').send_keys(username)
        t.sleep(2)
        self.driver.find_element_by_id('freepassword').
send_keys(password)
        t.sleep(2)
        self.driver.find_element_by_link_text('登录').click()

    def getLoginError(self):
        ''' 返回点击"登录"按钮后的错误提示信息'''
        t.sleep(2)
        loginError=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
')
        return loginError.text

@data(*readExcels())
@unpack
def test_sina_login(self,username,password,result):
    ''' 登录业务测试'''

```

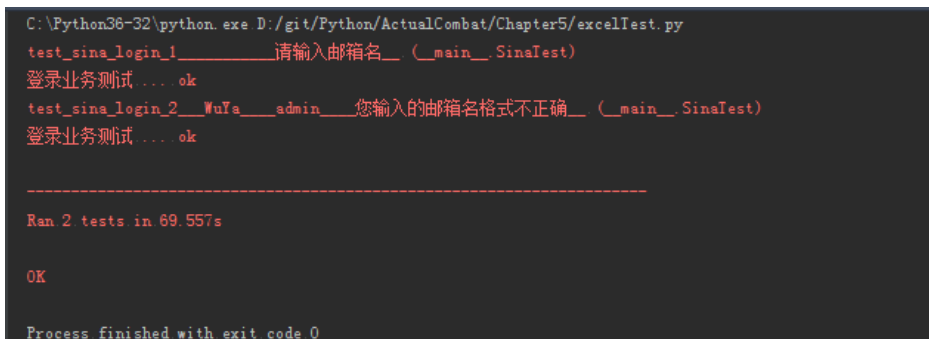
```

        self.login(username,password)
        self.assertEqual(self.getLoginError(),result)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：对读取 Excel 数据函数进行了重写，函数 readExcels 是读取 Excel 所有的数据并且把这些数据添加到 rows 列表中返回。接下来结合 ddt，在 data 函数 readExcels 前加了“*”号，这样数据类型变为元组，只需要写一个测试用例的代码就可以达到多个测试点的测试覆盖，执行后的结果如图 6-4-5 所示。



```

C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter5/excelTest.py
test_sina_login_1_____请输入邮箱名__ (_main__ SinaTest)
登录业务测试 ... ok
test_sina_login_2__WuYa____admin____您输入的邮箱名格式不正确__ (_main__ SinaTest)
登录业务测试 ... ok

-----

Ran 2 tests in 69.557s

OK

Process finished with exit code 0

```

图 6-4-5

在图 6-4-5 中可以很清晰地看到执行时的情况。

6.5 Xml 实战

Xml 文件是可扩展标记语言。在自动化测试中，也可以把数据存储到 Xml 文件中，这样应用时可直接从 Xml 文件中读取。

1. Xml 文件的读取

使用标准库 xml.dom.minidom，通过 document 的方式读取 Xml 文件的内容，在这里创建 sina.xml 文件，文件的内容为：

```

<?xml version="1.0" encoding="utf-8"?>
<DataDriven>
    <url>http://mail.sina.com.cn/</url>
    <errorMsg emailNull="请输入邮箱名" emailFormat="您输入的邮箱名格式不正确"
"></errorMsg>
</DataDriven>

```

下面读取 xml 文件中 url 节点中的内容和 errorMsg 节点中子节点的内容，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import xml.dom.minidom

def getXmlData(value):
    """
    获取 xml 单节点中的数据
    : param value:xml 文件中单节点的名称
    """
    dom = xml.dom.minidom.parse('sina.xml')
    db = dom.documentElement
    name = db.getElementsByTagName(value)
    nameValue = name[0]
    return nameValue.firstChild.data

def getXmlUser(parent, child):
    """
    获取 xml 子节点中的数据
    : param parent:xml 文件中父节点的名称
    : param child:xml 文件中子节点的名称
    """
    dom = xml.dom.minidom.parse('sina.xml')
    db = dom.documentElement
    itemlist = db.getElementsByTagName(parent)
    item = itemlist[0]
    return item.getAttribute(child)
```

注解：在以上代码中，对 Xml 文件的处理是通过 document 的形式来读取 Xml 文件中的内容，dom.documentElement 获得文档元素的结构来完成的。方法 getXmlUser 用来获取节点里面的数据，而方法 getXmlData 则获取单节点的数据。

2. Xml 在自动化测试中的实战

这里依然以新浪邮箱登录为例，测试当登录邮箱名为空和登录时填写的邮箱

格式不正确时，返回的错误提示信息，并把测试中用到的数据分离到 Xml 文件中，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya


import xml.dom.minidom
import unittest
from selenium import webdriver
import time as t


def getXmlData(value):
    '''
    获取 xml 单节点中的数据
    : param value:xml 文件中单节点的名称
    '''
    dom = xml.dom.minidom.parse('sina.xml')
    db = dom.documentElement
    name = db.getElementsByTagName(value)
    nameValue = name[0]
    return nameValue.firstChild.data


def getXmlUser(parent, child):
    '''
    获取 xml 子节点中的数据
    : param parent:xml 文件中父节点的名称
    : param child:xml 文件中子节点的名称
    '''
    dom = xml.dom.minidom.parse('sina.xml')
    db = dom.documentElement
    itemlist = db.getElementsByTagName(parent)
    item = itemlist[0]
    return item.getAttribute(child)


class SinaTest(unittest.TestCase):
    def setUp(self):
        self.driver=webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.get(getXmlData('url'))
        self.driver.implicitly_wait(30)
```

```

def tearDown(self):
    self.driver.quit()

def login(self, username, password):
    t.sleep(2)
    #邮箱账号输入框
    self.driver.find_element_by_id('freename').send_keys(username)
    t.sleep(2)
    self.driver.find_element_by_id('freepassword').
send_keys(password)
    t.sleep(2)
    self.driver.find_element_by_link_text('登录').click()

@property
def getLoginError(self):
    '''返回点击“登录”按钮后的错误提示信息'''
    t.sleep(2)
    loginError=self.driver.find_element_by_xpath('
/html/body/div[1]/div/div[2]/div/div/div[4]/div[1]/div[1]/div[1]/span[1]
')
    return loginError.text

def test_sina_login_emailNull(self):
    '''新浪邮箱登录: 登录账号邮箱为空验证'''
    self.login('', '')
    self.assertEqual(self.getLoginError,
getXmlUser('errorMsg', 'emailNull'))

def test_sina_login_emailFormat(self):
    '''新浪邮箱登录: 登录账号邮箱格式填写错误验证'''
    self.login('wuyal303', 'adminasd')
    self.assertEqual(self.getLoginError,
getXmlUser('errorMsg', 'emailFormat'))

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

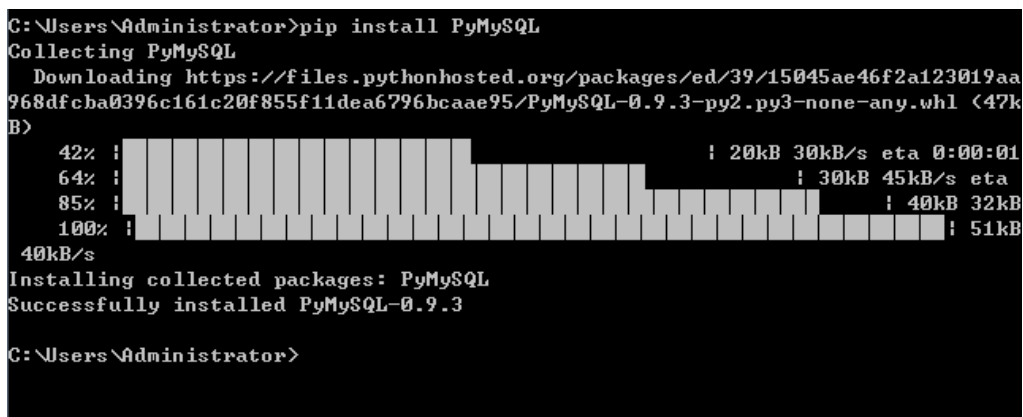
注解：在以上测试代码中，把数据分离到 xml 文件后，则登录的测试用例中不管是邮箱为空还是邮箱格式错误，直接从 xml 文件中读取数据与实际测试获取的数据进行断言比较。数据分离到 xml 后，即使错误提示信息进行了优化也可以直接在 xml 文件中修改。

6.6 MySQL 实战

在 Python2 中，对 MySQL 的操作使用的是第三方库 MySQLdb，很遗憾的是该库不支持 Python3。在 Python3 中对 MySQL 操作的库是 PyMySQL，该库需要单独进行安装，安装的命令为：^①

```
pip install PyMySQL
```

安装该库如图 6-6-1 所示。

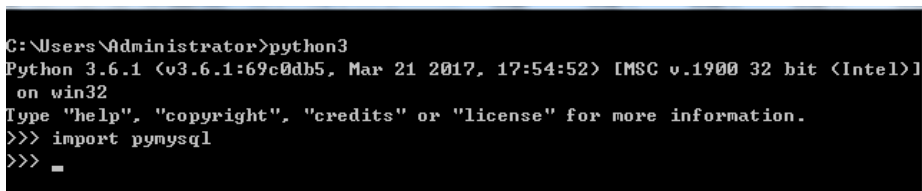


```
C:\Users\Administrator>pip install PyMySQL
Collecting PyMySQL
  Downloading https://files.pythonhosted.org/packages/ed/39/15045ae46f2a123019aa968dfc
ba0396c161c20f855f11dea6796bcaae95/PyMySQL-0.9.3-py2.py3-none-any.whl (47kB)
42% |
64% |
85% |
100% |
40kB/s
Installing collected packages: PyMySQL
Successfully installed PyMySQL-0.9.3

C:\Users\Administrator>
```

图 6-6-1

安装成功后，在 cmd 命令提示符下输入 python3 进入到 Python3 的环境中，输入 import pymysql，如无任何的错误提示表示该库安装成功，如图 6-6-2 所示。



```
C:\Users\Administrator>python3
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pymysql
>>> _
```

图 6-6-2

^① 在 Windows 的 cmd 命令提示符中，pip3 查询库或者安装库，都会忽略库的大小写。而在 Python 的正式编辑代码中，库则严格遵循大小写。在 Python 中，模块的名称一般都是小写，每一个 Python 文件就是一个模块，但是类的定义，首字母必须是大写。

在这里就不详细地介绍 MySQL 数据库的安装，假设 MySQL 的数据库已经安装好了。在 Windows 环境中启动 MySQL 服务，启动服务如图 6-6-3 所示。

```
C:\Users\Administrator>net start mysql
MySQL 服务正在启动。
MySQL 服务已经启动成功。

C:\Users\Administrator>
```

图 6-6-3

启动 MySQL 服务后，就可以进入到 MySQL 客户端进行操作，进入的命令为：

```
mysql -h localhost -u root -p
```

按下回车键后，输入 MySQL 账号 root 的密码，就可以进入到命令行操作 MySQL 了，如图 6-6-4 所示。

```
C:\Users\Administrator>mysql -h localhost -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.37 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

图 6-6-4

进入到客户端后，在数据库 db 中创建 user 表，并且插入一条数据，见创建的脚本和 db 数据库中 user 表的数据，如图 6-6-5 所示。

```

mysql> use db;
Database changed
mysql> show create table user \g;
+-----+
| Table | Create Table
+-----+
| user  | CREATE TABLE 'user' (
  'id' int(11) DEFAULT NULL,
  'username' varchar(20) DEFAULT NULL,
  'password' varchar(20) DEFAULT NULL,
  'nick' varchar(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ;
+-----+

1 row in set (0.00 sec)

ERROR:
No query specified

mysql> select * from user;
+-----+-----+-----+-----+
| id  | username | password | nick |
+-----+-----+-----+-----+
| 1   | wuya     | admin   | WuYa |
+-----+-----+-----+-----+

1 row in set (0.00 sec)

```

图 6-6-5

PyMySQL 库连接 MySQL 数据库的代码如下:

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import pymysql

def connMySQL():
    try:
        con=pymysql.connect(host='127.0.0.1',user='root',passwd='server',db='db')
    except:
        raise '连接 MySQL 数据库失败'
    else:pass
    finally:pass

connMySQL()

```

注解：在 PyMySQL 库中，连接 MySQL 的数据库直接调用 connect 函数，在该函数中需要指定连接 MySQL 数据库的 IP 地址、账号、账号密码和要操作的数据库。这里需要操作的数据库是 db。

MySQL 连接成功后，就可以实现对 MySQL 数据库的操作，这里先实现插入数据到 MySQL 的数据库，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import pymysql

def connMySQL():
    ''' 连接MySQL 数据库'''
    try:
        con=pymysql.connect(host='127.0.0.1',user='root',
        passwd='server',db='db')
        return con
    except:
        raise '连接 MySQL 数据库失败'

def insertOne():
    ''' 插入一条数据'''
    con=connMySQL()
    cur=con.cursor()
    sql='INSERT INTO USER VALUES (%s,%s,%s,%s)'
    params=(2,'weke','admin','WeKe')
    cur.execute(sql,params)
    con.commit()
    cur.close()
    con.close()

def insertMany():
    ''' 插入多条数据'''
    con=connMySQL()
    cur=con.cursor()
    sql='INSERT INTO USER VALUES (%s,%s,%s,%s)'
    params=[(3,'weike','admin','WeiKe'),
            (4,'lisi','admin','LiSi')]
    cur.executemany(sql,params)
```

```

con.commit()
cur.close()
con.close()

```

注解：在 PyMySQL 库中插入数据，有两种操作方式，一种是插入单条数据，另外一种就是批量插入数据，也就是多条数据的插入。多条语句数据的插入用到的方法是 `executemany`。另外需要注意的是，插入数据后，一定要运行 `commit`，否则系统只是执行了插入的操作，而数据并没有真正插入到数据库中。插入数据执行完成后，切记要关闭游标和连接对象。

执行插入函数后，数据就会插入到 db 数据库的 user 表中，如图 6-6-6 所示。

```

mysql> select * from user;
+-----+-----+-----+-----+
| id | username | password | nick |
+-----+-----+-----+-----+
| 1 | wuya | admin | WuYa |
| 2 | weke | admin | WeKe |
| 3 | weike | admin | WeiKe |
| 4 | lisi | admin | LiSi |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

图 6-6-6

下面来看对 MySQL 数据的查询。查询同样包括对单条数据的查询和数据的批量查询，下面是一个具体实现的代码实例：

```

# !/usr/bin/env python

# -*-coding:utf-8-*-

# author:wuya

import pymysql

def connMySQL():
    '''连接MySQL数据库'''
    try:
        con = pymysql.connect(
            host='127.0.0.1',
            user='root',
            passwd='server',

```

```

        db='db')
    return con
except:
    raise '连接 MySQL 数据库失败'

def get_one():
    ''' 查询单条数据'''
    try:
        con = connMySQL()
    except:
        raise '连接 MySQL 数据库失败'
    else:
        cur = con.cursor()
        sql = 'SELECT * FROM USER WHERE id=%s'
        params = (1,)
        cur.execute(sql, params)
        print('查询的结果:', cur.fetchone())
    finally:
        cur.close()
        con.close()

def get_many():
    ''' 批量查询数据'''
    try:
        con = connMySQL()
    except:
        raise '连接 MySQL 数据库失败'
    else:
        cur = con.cursor()
        sql = 'SELECT * FROM USER'
        cur.execute(sql)
        print('查询的结果:')
        for item in cur.fetchall():
            print(item)
    finally:
        cur.close()
        con.close()

```

注解：查询数据库后，如果是单条数据查询，查看查询结果使用的方法是 `fetchone`；多条数据查询则使用 `fetchall` 方法，并可以循环取出对应的数据。

下面使用 `MySQL` 的方法，来模拟一个系统登录的过程，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*

```

```

#author:wuya

import pymysql

class MySQLHelper(object):
    def conn(self):
        ''' 连接MySQL 数据库'''
        try:
            con = pymysql.connect(
                host='127.0.0.1',
                user='root',
                passwd='server',
                db='db')
            return con
        except:
            raise '连接 MySQL 数据库失败'

    def get_one(self,sql,params):
        try:
            cur=self.conn().cursor()
            cur.execute(sql,params)
            return cur.fetchone()
        except:pass
        finally:
            cur.close()
            self.conn().close()

class CheckUserInfo(MySQLHelper):
    def __init__(self):
        self.__helper=MySQLHelper()

    def checkUser(self,sql,params):
        return self.get_one(sql,params)

def userInfo():
    try:
        username = input('请输入账号:\n')
        passwd = input('请输入账号密码:\n')
        sql = 'SELECT * FROM USER WHERE username=%s and password=%s'
        params = (username, passwd)
        check = CheckUserInfo()
        result = check.checkUser(sql, params)
        nick = result[3]
        if result:

```

```

        print('登录系统成功，您的昵称为: {0}'.format(nick))
    else:
        print('您的账号或者用户名错误，请检查，谢谢! ')
except:
    raise '系统出错'

```

注解：在以上代码中，首先在类 `MySQLHelper` 中编写了连接 `MySQL` 和查询 `MySQL` 的方法，然后编写类 `CheckUserInfo`，在该类中检查输入的账号和密码是否在数据库中存在，最后编写 `userInfo` 函数进行验证。输入登录系统的账号和密码到数据库校验，如果用户存在则获取登录系统的账号昵称并打印出来；如果账号校验失败，直接打印出登录失败的错误信息。执行 `usrInfo` 函数后，执行结果如图 6-6-7 所示。

```

if __name__ == '__main__':
    userInfo()

```

C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter6/login.py
 请输入账号:
 wuYa
 请输入账号密码:
 123456
 登录系统成功，您的昵称为: WuYa

 Process finished with exit code 0

图 6-6-7

第 7 章 Page Objects 实战

在 UI 级的自动化测试框架中，怎样更高效地维护代码，特别是当页面样式改变或者页面元素属性改变，如何做到高效快速地修改代码来适应这些改变了？这个时候就可以考虑使用 Page Objects，也就是页面对象设计模式。在下面知识中结合具体的实例来说明页面对象设计模式在 UI 自动化测试中的应用。

7.1 Page Objects 的实现

在 UI 级的自动化测试中，页面对象设计模式表示测试正在交互的 Web 应用程序用户界面中的一个区域。这减少了代码的重复，也就是说，如果用户界面发生改变，只需要在一个地方修改程序就可以了，使用页面对象设计模式的优点为：

- （1）创建可以跨多个测试用例共享的代码；
- （2）减少重复代码的数量；
- （3）如果用户界面发生变更后，只需要在一个地方维护就可以了。

创建 UI，在 UI 的工程中创建对应的包和目录，如图 7-1-1 所示。



图 7-1-1

注解：在以上工程目录中，在 **base** 包里面存放基础代码，在 **page** 包里面编写关于页面对象层的代码，若 **Web** 页面发生变更，修改代码主要是在 **page** 包中进行。**utils** 包中编写读取文件的方法，**testCase** 包中编写页面对象中所有的测试代码，**data** 文件夹存放数据，测试数据存储在 **Xml** 文件中，**report** 存放测试报告。

在 **base** 包中创建 **basePage.py** 的模块，在该模块中编写基础代码，**basePage.py** 模块的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.expected_conditions import
NoSuchElementException
from selenium.webdriver.common.by import By
import time as t

class WebDriver(object):
    def __init__(self,driver):
        self.driver=driver

    def findElement(self,*loc):
        '''单个定位元素的方法'''
        try:
            return self.driver.find_element(*loc)
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))

    def findsElement(self,*loc):
        '''多个定位元素的方法'''
        try:
            return self.driver.find_elements(*loc)
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))
```

注解：在以上代码中，定义了类 **WebDriver**，类的构造函数中 **driver** 指的是 **webdriver** 实例化后对象，在 **WebDriver** 类中编写了单个定位元素的方法和多个定位元素的方法。不管是单个元素定位的方法还是多个元素定位的方法，参数都是 ***loc**，即可以识别元素属性，元素属性主要包含了 **ID**、**NAME**、


```

def typeUserName(self, username):
    self.findElement(*self.username_loc).send_keys(username)

def typePassword(self, password):
    self.findElement(*self.password_loc).send_keys(password)

@property
def clickLogin(self):
    self.findElement(*self.login_loc).click()

def login(self, username, password):
    self.typeUserName(username)
    self.typePassword(password)
    self.clickLogin

@property
def getLoginError(self):
    ''' 获取错误的信息'''
    return self.findElement(*self.loginError_loc).text

```

注解：以上代码定义了 Sina 的类，该类继承了 WebDriver 的类。在 Sina 类中，以类属性的方式指明每个操作的元素属性的值、Id、Name 等，然后依据每个操作步骤编写对应的方法。因为在以上实例中实现的是登录的操作，所以把登录的操作方法封装成一个 login 的方法，这样实现登录的测试用例直接调用 login 的方法。getLoginError 返回登录失败的信息，例如，账号和密码为空时点击“登录”按钮返回的错误提示信息。

接下来编写测试层的测试用例代码。在 testCase 包中创建 test_sina.py 文件，在该模块文件中编写测试用例，这里以账号和密码为空，以及账号格式不正确为测试点，test_sina.py 的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from page.sina import *
from selenium import webdriver
import time as t

class SinaTest(unittest.TestCase,Sina):

```

```

def setUp(self):
    self.driver=webdriver.Firefox()
    self.driver.maximize_window()
    self.driver.maximize_window()
    self.driver.get('https://mail.sina.com.cn/')

def tearDown(self):
    self.driver.quit()

def test_sinaLogin_001(self):
    ''' 登录业务: 账号密码为空验证 '''
    self.login('', '')
    self.assertEqual(self.getLoginError, '请输入邮箱名')

def test_sinaLogin_002(self):
    ''' 登录业务: 输入不规范邮箱名 '''
    self.login('wuyal303', 'asd888')
    self.assertEqual(self.getLoginError, '您输入的邮箱名格式不正确')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在测试模块 `test_sina.py` 的文件中，首先需要导入对象层的类 `Sina` 和 `unittest` 单元测试框架。在测试类 `SinaTest` 中，继承了 `unittest.TestCase` 和 `Sina` 类，继承 `TestCase` 是由于在编写自动化测试的用例中，用到的测试固件、测试断言和测试执行都需要 `TestCase` 类中的方法，而 `Sina` 类中所包含的对象层的测试操作步骤的方法，继承后可以直接进行调用。测试类 `SinaTest` 中编写了两个测试点，在编写测试用例时需要添加备注信息，明确注明该测试用例是测试哪个测试点，验证哪个场景，这样，在测试用例执行失败后，就可以很快知道是哪个业务的哪个功能出现了问题。

在这里把错误的信息和测试地址单独地分离出来，在 `data` 目录下创建 `ui.xml` 文件，在该文件中编写需要分离的数据，见 `ui.xml` 文件的内容：

```

<?xml version="1.0" encoding="utf-8"?>
<DataDriven>
    <url>https://mail.sina.com.cn/#</url>
    <divText emailNull="请输入邮箱名" emailFormat="您输入的邮箱名格式不正确"
"></divText>
</DataDriven>

```

在 `utils` 包中创建 `operationXml.py` 的模块，专门用于编写读取 `Xml` 文件的内容，`operationXml.py` 模块的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import os
import xml.dom.minidom

class OperationXml(object):
    def dir_base(self, fileName, filePath='data'):
        '''
        获取 data 文件夹下的文件
        : param fileName:要读的文件名称
        : param filePath:要读的文件名对应的文件夹
        '''
        return os.path.join(os.path.dirname(
            os.path.dirname(__file__)), filePath, fileName)

    def getXmlData(self, value):
        '''
        获取 xml 单节点中的数据
        : param value:xml 文件中单节点的名称
        '''
        dom = xml.dom.minidom.parse(self.dir_base('ui.xml'))
        db = dom.documentElement
        name = db.getElementsByTagName(value)
        nameValue = name[0]
        return nameValue.firstChild.data

    def getXmlUser(self, parent, child):
        '''
        获取 xml 子节点中的数据
        : param parent:xml 文件中父节点的名称
        : param child:xml 文件中子节点的名称
        '''
        dom = xml.dom.minidom.parse(self.dir_base('ui.xml'))
        db = dom.documentElement
        itemlist = db.getElementsByTagName(parent)
        item = itemlist[0]
        return item.getAttribute(child)
```

接下来重构刚才的测试代码。直接从 Xml 文件中读取 Url 和错误信息，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from page.sina import *
from page.init import *
import time as t
from utils.operationXml import *

class SinaTest(unittest.TestCase,Sina,OperationXml):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.maximize_window()
        self.driver.get(self.getXmlData('url'))

    def tearDown(self):
        self.driver.quit()

    def test_sinaLogin_001(self,parent='divText',
value='emailNull'):
        '''登录业务:账号密码为空验证'''
        self.login('', '')
        self.assertEqual(self.getLoginError,
self.getXmlUser(parent,value))

    def test_sinaLogin_002(self,parent='divText',
value='emailFormat'):
        '''登录业务:输入不规范邮箱名'''
        self.login('wuya1303','asd888')
        self.assertEqual(self.getLoginError,
self.getXmlUser(parent,value))

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

注解：通过以上方式就可以把测试 URL，返回的错误信息分离到 Xml 中，即使某些时候由于某些情况需要修改，只需要在 Xml 文件中修改对应的值就可以了。

在介绍 unittest 单元测试框架的时候介绍过，为了后期维护方便，不用重复

地编写测试固件，可以把测试固件进行分离，这样后期维护会更方便。例如，在 page 包中创建 init.py 文件，把测试固件单独分离出来，init.py 的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from selenium import webdriver
from utils.operationXml import *

class Init(unittest.TestCase,OperationXml):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.maximize_window()
        self.driver.get('https://mail.sina.com.cn/')

    def tearDown(self):
        self.driver.quit()
```

分离出测试固件后，那么，测试层中的 SinaTest 测试类只需要继承 init 模块中的 Init 类就可以了，test_sina.py 模块修改后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from page.sina import *
from page.init import *
import time as t

class SinaTest(Init,Sina):

    def test_sinaLogin_001(self,parent='divText',value='emailNull'):
        ''' 登录业务: 账号密码为空验证'''
        self.login('', '')
        self.assertEqual(self.getLoginError,
self.getXmlUser(parent,value))

    def test_sinaLogin_002(self,parent='divText',value='emailFormat'):
        ''' 登录业务: 输入不规范邮箱名'''
```



```

        self.login('wuyal303','asd888')
        self.assertEqual(self.getLoginError,
self.getXmlUser(parent,value))

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

为了批量执行所有的测试用例，可在 `allTests.py` 模块中编写执行所有测试用例及生成测试报告的方法，`allTests.py` 模块的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import os
import HTMLTestRunner
import time

def allTests():
    ''' 获取所有需要执行的测试用例'''
    suite=unittest.defaultTestLoader.discover(
        start_dir=os.path.join(os.path.dirname(__file__),'testCase'),
        pattern='test_*.py',
        top_level_dir=None
    )
    return suite

def getNowTime():
    ''' 获取当前的时间'''
    return time.strftime('%Y-%m-%d %H_%M_%S',time.localtime(time.time()))

def run():
    fileName=os.path.join(os.path.dirname(__file__),
'report',getNowTime()+'.sinaReport.html')
    fp=open(fileName,'wb')
    runner=HTMLTestRunner.HTMLTestRunner(
        stream=fp,
        title='UI 自动化测试报告',
        description='UI 自动化测试报告详细信息')
    runner.run(allTests())

```

```
if __name__ == '__main__':  
    run()
```

注解：运行以上代码后，系统就会执行所有的测试用例，执行完成后生成的测试报告存储在 **report** 的目录下。

经过以上完善，最新的目录如图 7-1-4 所示。

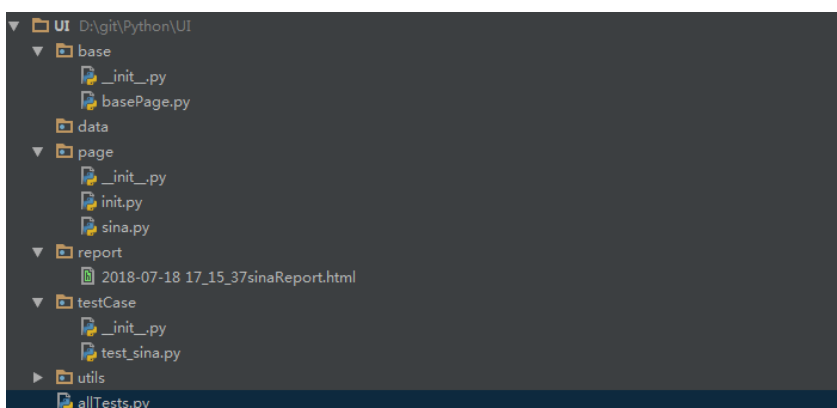


图 7-1-4

7.2 Page Objects 中引入 wait

在 7.1 小节介绍的 **base** 包里 **basePage.py** 模块中的基础代码，并不是最佳的方式，在该代码中加入显式等待来解决判断元素是否存在后再进行如点击，输入等操作，在 **basePage.py** 模块的代码中引入 **WebDriverWait** 类，完善后的代码如下：

```
#!/usr/bin/env python  
#-*-coding:utf-8-*-  
  
#author:wuya  
  
from selenium import webdriver  
from selenium.webdriver.support.expected_conditions import  
NoSuchElementException  
from selenium.webdriver.support.wait import WebDriverWait  
from selenium.webdriver.common.by import By
```

```

import time as t

class WebDriver(object):
    def __init__(self,driver):
        self.driver=driver

    def findElement(self,*loc):
        '''单个定位元素的方法'''
        try:
            return WebDriverWait(self.driver,20).until(lambda
x:x.find_element(*loc))
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))

    def findsElement(self,*loc):
        '''多个定位元素的方法'''
        try:
            return WebDriverWait(self.driver,20).until(lambda
x:x.find_elements(*loc))
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))

```

注解：以上代码在原来的基础上增加了显式等待，设置的时间为 20 s，即允许系统网络不稳定的情况下最大等待的时间是 20 s。lambda 是 python 的匿名函数，在 lambda 函数中，左边的是匿名函数的参数，右边的是匿名函数的表达式。例如，实现两个数相加的函数 add，参数是 a，b，那么调用 add 函数赋予实际参数可以实现两个数的相加，使用 lambda 实现起来更加简单，lambda 实例的代码如图 7-2-1 所示。

```

Microsoft Windows [版本 6.1.7601
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>python3
Python 3.6.1 <v3.6.1:69c0db5, Mar 21 2017, 17:54:52> [MSC v.1900 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> def add(a,b):
...     return a+b
...
>>> add(2,3)
5
>>> c=lambda a,b:a+b
>>> c(2,3)
5
>>>

```

图 7-2-1

再次执行 `testCase` 模块下的 `test_sina.py` 中测试类中的测试代码，验证本次修改基础代码是否对测试用例有影响，运行代码后测试用例全部通过。

7.3 Page Objects 引入工厂设计模式

在 UI 级别的自动化测试中，Selenium 主要应用在 Web 应用程序的自动化测试中，Appium 应用在 App 应用程序的自动化测试中，在 Appium 的自动化测试框架中，元素定位的类继承了 Selenium 代码中的 `By` 类，`mobileby` 模块下的 `MobileBy` 类的代码如下：

```
from selenium.webdriver.common.by import By

class MobileBy(By):
    IOS_PREDICATE = '-ios predicate string'
    IOS_UIAUTOMATION = '-ios uiautomation'
    IOS_CLASS_CHAIN = '-ios class chain'
    ANDROID_UIAUTOMATOR = '-android uiautomator'
    ACCESSIBILITY_ID = 'accessibility id'
    IMAGE = '-image'
```

在以上代码中可以看到，`MobileBy` 类继承了 `By` 类，也就是说在 App 的应用自动化测试中，元素定位也是支持元素属性 ID 等的。

对 `base` 模块下的 `basePage.py` 模块代码再次进行扩展，让元素定位部分的方法同时支持对 Web 应用程序产品的元素定位，也支持对 App 应用程序的元素定位。在这里引入工厂方法模式，在工厂方法模式中，工厂方法用于创建产品，并隐藏了产品对象实例化的过程。对 `basePage.py` 模块的代码再次进行重构，定义 `Factory` 类创建不同的测试对象，`basePage.py` 模块的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8 -*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.support.expected_conditions import \
NoSuchElementException
from selenium.webdriver.support.wait import WebDriverWait
```

```

from selenium.webdriver.common.by import By
from appium.webdriver.common.mobileby import MobileBy
import time as t

class Factory(object):
    def __init__(self,driver):
        self.driver=driver

    #工厂方法
    def createDriver(self,driver):
        if driver=='web':
            return WebUI(self.driver)
        elif driver=='app':
            return AppUI(self.driver)

class WebDriver(object):
    def __init__(self,driver):
        self.driver=driver

    def findElement(self,*loc):
        ''' 单个定位元素的方法'''
        try:
            return WebDriverWait(self.driver,20).until(lambda
x:x.find_element(*loc))
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))

    def findsElement(self,*loc):
        ''' 多个定位元素的方法'''
        try:
            return WebDriverWait(self.driver,20).until(lambda
x:x.find_elements(*loc))
        except NoSuchElementException as e:
            print('Error Details {0}'.format(e.args[0]))

class WebUI(WebDriver):
    def __str__(self):
        return 'WebUI'

class AppUI(WebDriver):
    def __str__(self):
        return 'AppUI'

```

注解：在上面的代码中，在 Factory 类中定义了工厂类，Factory 类生成

WebDriver 对象。定义 Factory 类创建不同的 WebDriver 对象。WebUI 类和 AppUI 类继承自 WebDriver 类，WebUI 和 AppUI 可以看作是具体的测试对象产品（Web 和 App）。在 Factory 类中定义了工厂方法 createDriver，工具字符串类型 driver 的值，生成不同的 WebDriver 对象。如果 driver 对象是“web”，则调用 WebUI，返回 WebUI 类的实例。如果 driver 对象是“app”，则调用 AppUI，返回 AppUI 类的实例。

重构基础代码后，在对象层的代码中，如果是基于 Web 应用程序的类就继承 WebUI，如果是基于 App 应用程序的类就继承 AppUI，在前面的 page 模块下的 sina.py 模块中，对象层 Sina 类由原来继承 WebDriver 修改为继承 WebUI，再次执行 testCase 模块下 test_sina.py 模块中的测试代码，可以正常地执行通过。

在以上实例中，引入了工厂设计模式，这里以微博的 App 作为测试实例来说明该模式在 UI 自动化测试中的应用，不管是针对 WEB 还是 App。在对象层的包中创建 weibo.py 的模块，在该模块中创建 WeiBo 类，该类继承 AppUI，在类 WeiBo 中编写微博获取手机验证码的方法，weibo.py 模块的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from selenium import webdriver
from selenium.webdriver.common.by import By
from base.basePage import *
from time import sleep
from appium import webdriver

class WeiBo(AppUI):
    login_loc=(By.XPATH,"//android.widget.TextView[@text='登录']")
    phone_loc=(By.XPATH,"//android.widget.EditText[@text='输入手机号']")
    codeButton_loc=(By.XPATH,"//android.widget.Button[@text='获取验证码']")

    @property
    def clickMy(self):
        sleep(3)
        self.findElement(*self.login_loc).click()

    def typePhone(self,phone):
```

```

        self.findElement(*self.phone_loc).send_keys(phone)

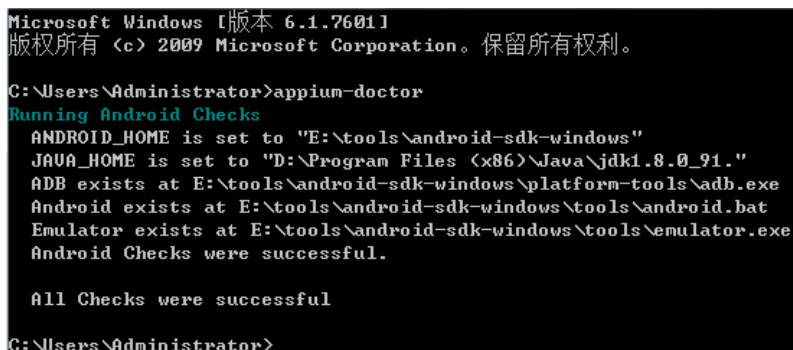
    def clickCodeButton(self):
        self.findElement(*self.codeButton_loc).click()

    def getPhoneCode(self, phone):
        self.clickMy
        self.typePhone(phone)
        self.clickCodeButton()

```

注解：以上代码中，类 `WeiBo` 继承了 `AppUI`，在该类中首先获取元素的属性，接下来依次根据元素的属性编写对应的方法，最后把获取手机验证码的操作步骤封装成一个方法 `getPhoneCode`。

在测试包 `testCase` 中创建 `test_weibo`，编写对应的测试用例。在执行测试用例前，首先保证 `Appium` 的环境搭建是正确的，在 `cmd` 命令提示符下输入 `appium-doctor`，会自动检测 `Appium` 的环境是否正确，显示如图 7-3-1 所示。



```

Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>appium-doctor
Running Android Checks
  ANDROID_HOME is set to "E:\tools\android-sdk-windows"
  JAVA_HOME is set to "D:\Program Files (x86)\Java\jdk1.8.0_91."
  ADB exists at E:\tools\android-sdk-windows\platform-tools\adb.exe
  Android exists at E:\tools\android-sdk-windows\tools\android.bat
  Emulator exists at E:\tools\android-sdk-windows\tools\emulator.exe
  Android Checks were successful.

  All Checks were successful

C:\Users\Administrator>

```

图 7-3-1

编写测试代码，`test_weibo.py` 模块的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import time as t
from appium import webdriver
from page.weibo import WeiBo

```

```

class WeiBoTest(unittest.TestCase,WeiBo):
    def setUp(self):
        desired_caps={}
        desired_caps['platformName']='Android'
        desired_caps['platformVersion']='6.0'
        desired_caps['deviceName']='meizu_M5'
        desired_caps['appPackage']='com.sina.weibo'
        desired_caps['appActivity']='com.sina.weibo.SplashActivity'
        desired_caps['unicodeKeyboard']=True
        desired_caps['resetKeyboard']=True
        self.driver=webdriver.Remote('http://127.0.0.1:4723/wd/hub',
desired_caps)

    def test_001(self):
        t.sleep(3)
        self.getPhoneCode('13456787654')

    def tearDown(self):
        self.driver.quit()

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在测试模块 `test_weibo.py` 中首先需要导入 `unittest` 单元测试框架、`appium` 包中对应的 `webdriver` 类，以及对象层的代码 `WeiBo` 类。在测试类 `WeiBoTest` 中继承了 `TestCase` 和 `WeiBo` 类之后，接着在该测试类中编写获取手机验证码的测试用例。

接下来启动 Appium 的服务，启动后的 Appium 服务如图 7-3-2 所示：

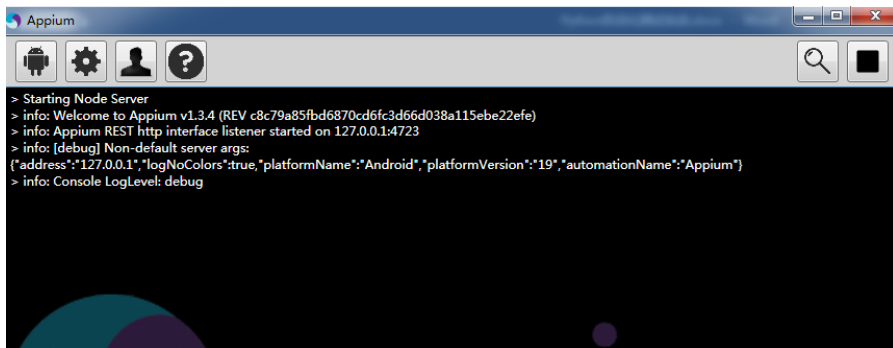


图 7-3-2

使用数据线在电脑上连接手机，并将手机设置为开发者模式，连接成功后，在电脑 cmd 命令提示符下输入 `adb devices -l` 终端显示如下的信息，表示手机连接成功，如图 7-3-3 所示。

```
C:\Users\Administrator>adb devices -l
List of devices attached
611AKBPK22XRJ      device product:meizu_M5 model:M5 device:M5

C:\Users\Administrator>_
```

图 7-3-3

运行 `test_weibo.py` 测试模块中的代码，测试可以正常执行。在移动产品的测试中，手机 `app` 启动后需要左右滑动才可以看到登录的页面，在 `base` 包中编写的 `swipe.py` 模块中对 `Swipe` 类进行了二次封装，封装的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

from appium import webdriver
import time as t

class Swipe(object):
    def __init__(self,driver):
        self.driver=driver

    @property
    def width(self):
        return self.driver.get_window_size()['width']

    @property
    def height(self):
        return self.driver.get_window_size()['height']

    @property
    def getResolution(self):
        return str(self.width)+"*"+str(self.height)

    @property
    def set_Left_Right(self):
        '''
```

```

        :return: 实现从左到右滑动,滑动时x 轴起点大于终点
        '''
        t.sleep(2)
        self.driver.swipe(self.width*9/10,self.height/2,
self.width/20,self.height/2,0)

    @property
    def set_Right_Left(self):
        '''
        :return: 实现从右到左滑动,滑动时x 轴起点小于终点
        '''
        t.sleep(2)
        self.driver.swipe(self.width/10,self.height/2,
self.width*9/10,self.height/2,0)

    @property
    def set_Up_Down(self):
        '''
        :return: 实现从上往下滑动,滑动时y 轴起点大于终点
        '''
        t.sleep(2)
        self.driver.swipe(self.width/2,self.height*9/10,
self.width/2,self.height/20,0)

    @property
    def set_Down_Up(self):
        '''
        :return: 实现从下往上滑动,滑动时y 轴起点小于终点
        '''
        t.sleep(2)
        self.driver.swipe(self.width/2,self.height/20,
self.width/2,self.height*9/10,0)

```

注解：Swipe 类的构造函数是 driver，实际上对 Swipe 类实例化的时候，它的参数就是 webdriver 实例化后的对象，例如，对 Swipe 类实例化，实例代码为 per=Swipe(self.driver)。接下来依次获取手机的高度和宽度，然后不管是上下滑动还是左右滑动都是根据手机坐标来进行设置和操作的。

结合以上实例，可以看到在框架中引入工厂设计模式后，对代码进行了重构和封装。框架既可以对 Web 应用产品做 UI 自动化测试，也可以对 App 产品做 UI 自动化测试。

第 8 章 UI 自动化测试实战

前面章节详细地介绍了 Selenium 相关的知识，本章将结合具体的实例，介绍 Selenium 在 UI 自动化测试中的实战应用。

8.1 Web 产品的实战

结合一个具体的产品，测试在一个用户登录系统后，创建一个用户，然后查询该用户，最后删除该用户。在第六章代码的基础上，在对象层的包中创建 login 的模块文件，在该模块中编写登录方法，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from base.basePage import *
from selenium.webdriver.common.by import By

class Login(WebUI):
    username_loc=(By.NAME,'username')
    password_loc=(By.NAME,'password')
    login_loc=(By.CSS_SELECTOR,'//*[@id="app"]/div/div/div[2]/form/div[4]/button')

    def typeUserName(self,username):
        self.findElement(*self.username_loc).send_keys(username)

    def typePassword(self,password):
        self.findElement(self.password_loc).send_keys(password)

    @property
    def clickLogin(self):
        self.findElement(*self.login_loc).click()
```

```
def login(self,username='66**66',passwd='asd***'):
    self.typeUserName(username)
    self.typePassword(passwd)
    self.clickLogin
```

接下来创建 user.py 模块的代码，在 user.py 模块中编写添加用户、查询用户和删除用户对象层的代码，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from base.basePage import *
from selenium.webdriver.common.by import By

class User(WebUI):
    #添加用户元素属性
    addUser_loc=(By.LINK_TEXT,'添加商户')
    #账号输入框元素属性
    account_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div/div[2]/div/div/div[1]/div/input')
    #用户姓名输入框元素属性
    name_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div/div[2]/div/div/div[2]/div/input')
    #账号密码输入框元素属性
    passwd_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div/div[2]/div/div/div[3]/div/input')
    #添加用户保存按钮元素属性
    save_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div/div[2]/div/div/div[10]/div/button')

    @property
    def clickAddUser(self):
        '''点击添加商户按钮'''
        self.findElement(*self.addUser_loc).click()

    def typeAccount(self,account):
        self.findElement(*self.account_loc).send_keys(account)

    def typeName(self,name):
        self.findElement(*self.name_loc).send_keys(name)

    def getUserName(self):
```

```

''' 获取用户输入框填写的用户名称'''
return self.findElement(*self.name_loc).get_attribute('value')

def typePasswd(self,passwd):
    self.findElement(*self.passwd_loc).send_keys(passwd)

@property
def clickSave(self):
    self.findElement(*self.save_loc).click()

def addUer(self,account='666666',name='无涯',passwd='123456'):
    '''创建用户'''
    self.clickAddUser
    self.typeAccount(account)
    self.typeName(name)
    name=self.getUserName()
    self.typePasswd(passwd)
    self.clickSave
    self.clickUserManage
    return name

'''用户列表'''
#导航栏用户管理元素属性
userManage_loc=(By.LINK_TEXT,'商户管理')
#用户列表中用户名称元素属性
userName_loc=(By.LINK_TEXT,'无涯')
#用户管理中用户查询输入框元素属性
userSo_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div[1]/div/div[2]
/input')
#查询按钮元素属性
userQuery_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div[1]/div
/div[3]/button')
#用户列表下拉框操作元素属性
userSel_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div[2]/div[2]
/div/table/tbody/tr/td[6]/div/i')
#删除用户元素属性
userDel_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div[2]/div[2]
/div/table/tbody/tr/td[6]/div/ul/li[2]')
#删除用户确定弹出框元素属性
userDelOk_loc=(By.XPATH,'/html/body/div[5]/div[2]/div/div/div
/div/div[2]/button[1]')
#用户列表无数据时的元素属性
listNo_loc=(By.XPATH,'//*[@id="app"]/div/div[1]/div[2]/div[2]
/div/table/tbody/tr/td/div')

```

```

@property
def clickUserManage(self):
    '''点击用户管理'''
    t.sleep(3)
    self.findElement(*self.userManage_loc).click()

@property
def getListUserName(self):
    '''获取用户列表中的用户名称'''
    return self.findElement(*self.userName_loc).text

def typeUserSo(self, name='无涯'):
    self.findElement(*self.userSo_loc).send_keys(name)

@property
def clickQuery(self):
    self.findElement(*self.userQuery_loc).click()

@property
def clickUserSel(self):
    '''点击用户列表操作下拉框'''
    self.findElement(*self.userSel_loc).click()

@property
def clickUserDel(self):
    '''点击删除用户按钮'''
    self.findElement(*self.userDel_loc).click()
    self.clickUserDelOk

@property
def clickUserDelOk(self):
    '''点击删除用户弹出框确定按钮'''
    t.sleep(2)
    self.findElement(*self.userDelOk_loc).click()

@property
def userDel(self):
    '''删除用户'''
    self.clickUserSel
    self.clickUserDel
    t.sleep(3)

@property
def getListNo(self):

```

```

''' 获取用户列表查询无数据时的提示信息'''
return self.findElement(*self.listNo_loc).text

def isAddUser(self):
    '''
    创建用户增加判断，如用户存在，先删除再创建
    如不存在，就直接创建用户
    '''
    try:
        self.typeUserSo()
        self.clickQuery
        assert self.getListUserName in '无涯'
        t.sleep(3)
        self.userDel
    except:
        return self.addUser()
    else:
        return self.addUser()
    finally:pass

```

注解：在 User 类中，编写了添加用户、用户列表、用户查询及删除用户的元素属性和单独的方法。在添加 addUser 用户的方法中，添加用户成功后返回添加用户时填写的用户名称，这样做的目的是在添加用户成功后，获取用户列表的用户名称，与添加用户时填写的用户名称进行断言，如果一致，表示添加用户成功，不一致表示添加用户存在问题。在添加用户的时候，获取用户名输入框填写的名称方法为 getUsername，此方法使用了 get_attribute(value) 获取输入框填写的内容。在添加用户时存在的问题是添加的用户已存在，从而导致添加用户失败，这并不能说明功能存在问题，而是在对象层中需要处理这类逻辑情况。在添加用户的时候，对添加的用户进行查询，如果用户存在（可能是手工测试添加的或者是上一次执行自动化测试代码添加的），则删除用户再次创建；如果不存在则创建用户。在对象层中该方法为 isAddUser，这样执行创建用户的测试用例，不管以后用户是否存在，代码都能够执行下去，如果不这样处理，可能因为对象层的代码没进行逻辑处理而导致代码执行失败，也就无法判断到底是添加用户的功能存在问题还是代码自身有问题。

在对象层的 init.py 模块中，在分离出的测试固件中将测试地址从新浪邮箱修改为 http://118.***.***.145:9098/#/login，修改后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from selenium import webdriver

class Init(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.maximize_window()
        self.driver.get('http://118.***.***.145:9999/#/login')

    def tearDown(self):
        self.driver.quit()
```

编写完对象层的代码后，在 `testCase` 模块下创建 `test_user.py` 的模块编写创建用户，查询用户和删除用户的测试用例，`test_user.py` 的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from page.user import *
from page.init import *
from page.login import *

class UserTest(Init,User):
    def login(self):
        '''登录系统的方法'''
        ecp=Login(self.driver)
        ecp.login()

    def test_user_add(self):
        '''用户管理业务:创建用户'''
        self.login()
        name=self.isAddUser()
        listName=self.getListUserName
        self.userDel
        self.assertEqual(listName,name)
```



```

def test_user_query(self):
    '''用户管理业务:查询用户'''
    self.login()
    name = self.isAddUser()
    #依据用户名查询用户
    self.typeUserSo()
    self.clickQuery
    listName=self.getListUserName
    self.userDel
    self.assertEqual(listName,name)

def test_user_delete(self):
    '''用户管理业务:删除用户'''
    self.login()
    #创建用户
    self.isAddUser()
    #删除用户
    self.userDel
    #依据用户名查询用户是否存在
    self.typeUserSo()
    self.clickQuery
    self.assertEqual(self.getListNo,'没有记录')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在 UserTest 类中，继承了 Init 类和 User 类，在 Init 类中编写了测试固件及需要测试的地址，而 User 类就是对象层的代码。在对象层 User 类中编写了添加用户、查询用户和删除用户的方法；在 UserTest 类中编写了添加用户、查询用户和删除用户的测试用例。由于在 UserTest 测试类中，不管是添加用户、查询用户还是删除用户，都需要登录到系统中，所以，可以把登录方法分离出来，单独写一个 login 方法。在 login 方法中，对 Login 实例化后调用了 Login 类中的 login 方法。这样，在添加用户、查询用户及删除用户时都可直接调用 login 方法登录到系统。在测试类 UserTest 中，添加用户，查询用户，删除用户的测试用例之间是没有任何依赖关系的，这样测试用例执行的时候就不会相互影响。不管是查询用户还是删除用户，首先都需要添加用户，每个测试用例执行完成后，之前添加的数据都会被清空，这样，在下次执行的时候就不会有影响。在添加用户测试用例，也就是 test_user_add 的测试用例中，添加用户成功后，先获取用户填写的用户名称，最后获取添加用户成功后用户列表中显示的用户名称，再删除用户，

最后对添加的测试用例进行断言。这里特别需要注意的是，断言务必放在删除用户之后，不管添加用户这个测试用例是否执行成功，添加用户后，把这条数据删除，再来验证添加用户填写的用户名称和用户列表中显示的用户名称。如果断言在删除用户之前，有可能断言失败，导致删除用户的代码无法执行，对下一次执行测试用例产生影响。

在 `init.py` 模块中的 `Init` 类中，测试地址并没有分离出去，这里把测试中用到的数据分离到 `xml` 文件中，在 `data` 文件夹中创建 `ui.xml` 文件，内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<DataDriven>
    <url>http://118.***.***.145:9999/#/login</url>
</DataDriven>
```

在 `utils` 模块中创建 `helper.py` 模块，在 `helper.py` 模块中实现读取 `Xml` 文件中的内容，`helper.py` 的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import os
import xml.dom.minidom

class Helper(object):
    def dir_base(self,fileName,filePath='data'):
        '''
        获取 data 文件夹下的文件
        : param fileName:要读的文件名称
        : param filePath:要读的文件名对应的文件夹
        '''
        return os.path.join(os.path.dirname(
            os.path.dirname(__file__)),filePath,fileName)

    def getXmlData(self,value):
        '''
        获取 xml 单节点中的数据
        : param value:xml 文件中单节点的名称
        '''
        dom = xml.dom.minidom.parse(self.dir_base('ui.xml'))
        db = dom.documentElement
```

```

name = db.getElementsByTagName(value)
nameValue = name[0]
return nameValue.firstChild.data

def getXmlUser(self, parent, child):
    '''
    获取 xml 子节点中的数据
    : param parent:xml 文件中父节点的名称
    : param child:xml 文件中子节点的名称
    '''
    dom = xml.dom.minidom.parse(self.dir_base('ui.xml'))
    db = dom.documentElement
    itemlist = db.getElementsByTagName(parent)
    item = itemlist[0]
    return item.getAttribute(child)

```

注解：在 Helper 类中的 dir_base 方法用来获取需要读取文件夹下文件的路径，方法 getXmlData 用来读取 Xml 文件中单节点中的数据，方法 getXmlUser 用来读取父节点下子节点中的数据。

utils 包中的 helper 模块中的 Helper 类已实现读取 Xml 文件中的内容，这里把 init.py 模块中的 Init 类的代码进行重构，直接读取 Xml 文件中测试的地址，修改后的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
from selenium import webdriver
from utils.helper import *

class Init(unittest.TestCase,Helper):
    def setUp(self):
        self.driver = webdriver.Firefox()
        self.driver.maximize_window()
        self.driver.maximize_window()
        self.driver.get(self.getXmlData('url'))

    def tearDown(self):
        self.driver.quit()

```

注解：在 `init` 模块中先导入 `Helper` 类，使 `Init` 类继承 `Helper` 类。加载测试地址直接调用 `Xml` 文件中的测试地址。

在测试类 `UserTest` 删除用户的测试用例中，把提示信息“没有记录”分离到 `Xml` 并且读取它，该测试类的代码如下：

```
def test_user_delete(self, parent='data', child='notData'):
    ''' 用户管理业务: 删除用户 '''
    self.login()
    # 创建用户
    self.isAddUser()
    # 删除用户
    self.userDel
    # 依据用户名查询用户是否存在
    self.typeUserSo()
    self.clickQuery
    self.assertEqual(self.getListNo, self.getXmlUser(parent, child))
```

注解：在删除用户的测试用例代码中，把提示信息分离到 `Xml` 文件中，这样，在测试用例中就不会存在提示信息了，因为提示信息在多个地方会使用到，如需修改时，我们只需要在 `Xml` 文件中进行修改，而不需要在多个测试用例中应用到的地方逐一修改。

运行 `allTests.py` 模块的文件。测试执行成功后，会在 `report` 文件夹下生成测试报告，生成的测试报告如图 8-1-1 所示。

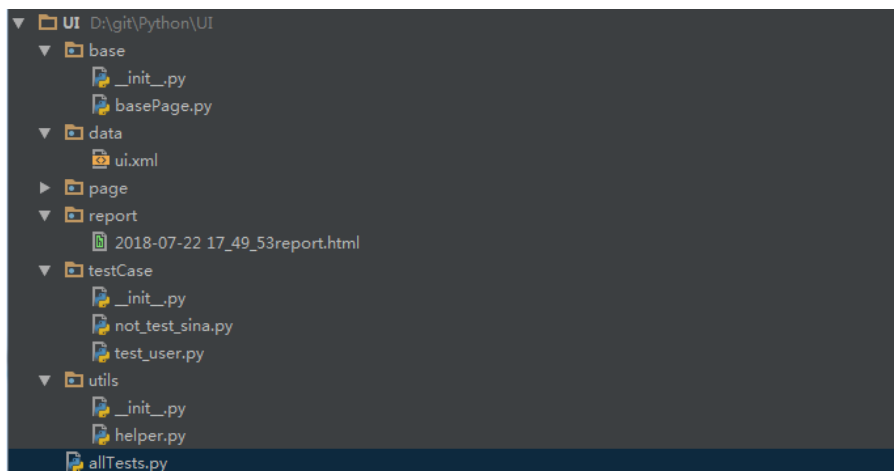


图 8-1-1

打开 report 文件夹下的测试报告，如图 8-1-2 所示。

← → ↻ | localhost:63342/rainui/report/2018-07-22%2017_49_53report.html

UI自动化测试报告

Start Time: 2018-07-22 17:49:53
Duration: 0:03:09.965865
Status: Pass 3

UI自动化测试报告详细信息

Show [Summary](#) [Failed](#) [All](#)

Test Group/Test case	Count	Pass	Fail	Error	View
test_user.UserTest	3	3	0	0	Detail
test_user_add: 用户管理业务:创建用户			pass		
test_user_delete: 用户管理业务:删除用户			pass		
test_user_query: 用户管理业务:查询用户			pass		
Total	3	3	0	0	

图 8-1-2

在 Jenkins 创建 Job，Job 名称为 UI，在 jenkins 执行 UI 工程下的测试用例，创建后的 Job 视图如图 8-1-3 所示。



图 8-1-3

选择“立即构建”选项后，执行测试用例，执行后的测试报告如图 8-1-4 所示。

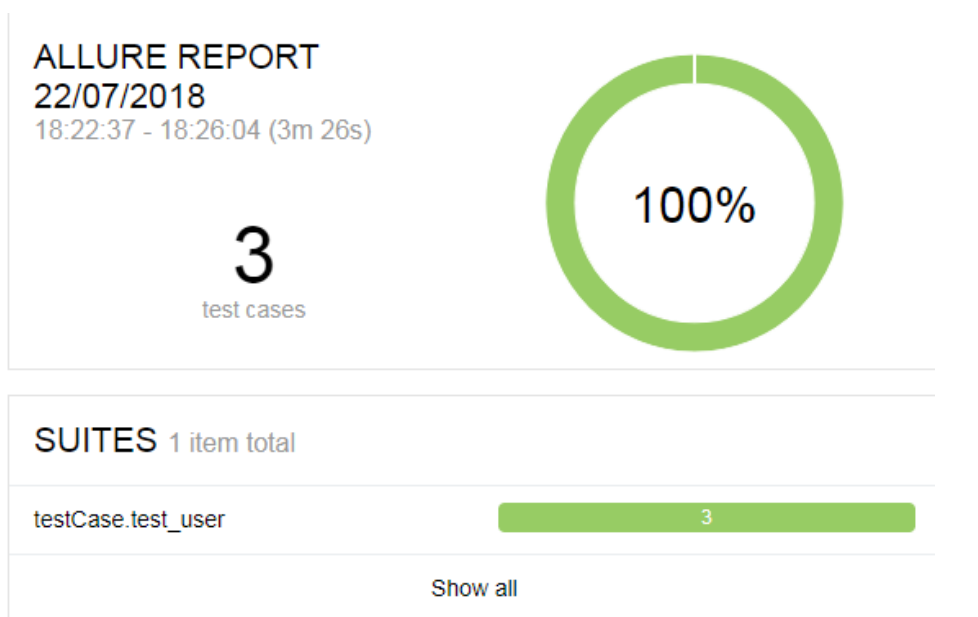


图 8-1-4

至此，一个完整的 UI 级的自动化测试代码更新完毕。在 UI 级的自动化测试中，最大的困难在于维护成本的增加，所以一定要将测试中的公共数据分离出来，并对用到的方法依据业务逻辑进行封装。



第2部分 接口自动化测试

第9章 HTTP协议

第10章 序列化与反序列化

第11章 PostMan的应用

第12章 JMeter接口测试应用

第13章 Requests实战

第9章 HTTP 协议

9.1 HTTP 简述

HTTP (HyperText Transfer Protocol) 协议是基于应用层的协议，完成客户端发送请求到服务端等一系列的运作流程，中文简称超文本传输协议。HTTP 诞生于 1989 年 3 月，最初由蒂姆·伯纳斯-李博士提出，目的是实现让远隔重洋的研究者们共享知识的设想。1997 年 1 月发布了 HTTP/1.1 的版本，也就是目前，比较主流的 HTTP 协议版本。由于 HTTP 是基于应用层的协议，因此客户端不需要关注底层的网络细节，连接请求由可靠的传输协议 TCP/IP 负责。当客户端发送请求给服务端的时候，客户端与服务器之间建立了通信，请求完成后客户端与服务端之间的通信完成，请求流程如图 9-1-1 所示。

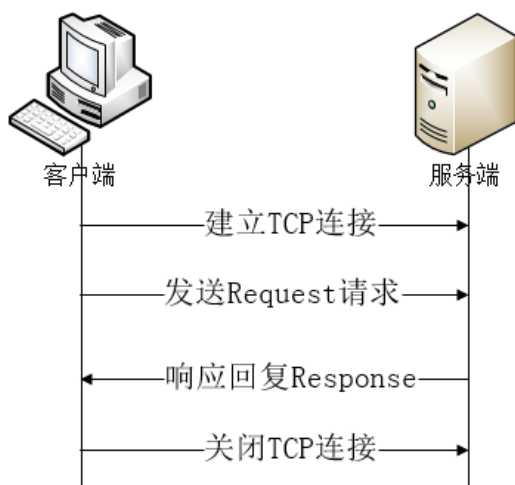


图 9-1-1

具体步骤为：

- (1) 在发送请求前，客户端与服务端之间请求建立通信，打开 TCP 的连接；

- (2) 建立 TCP 连接后，客户端发送请求到服务端；
- (3) 服务端收到响应后回复给客户端；
- (4) 客户端收到服务端的回应后关闭 TCP 的连接。

例如，访问百度首页，打开浏览器，输入 `https://www.baidu.com/` 域名后，按下回车键就可以跳转到百度的首页。在一个完整的 HTTP 请求流程中，客户端发送 HTTP 的请求给服务端，这中间需要明确地告诉服务端请求的地址，也就是统一资源定位符 URL，URL 主要由三部分组成，分别是 HTTP 协议，服务器地址和资源，例如，地址 `http://www.cnblogs.com/weke/category/831885.html` 中，`http://` 是协议，`www.cnblogs.com` 是被请求的服务器，`/weke/category/831885.html` 是请求资源。

在 HTTP 的协议中，客户端发送请求到服务端的过程中，客户端需要告诉服务端是以什么样的请求方式去请求。在 HTTP 的协议中，HTTP 的请求方法主要为 GET, POST, HEAD, PUT, DELETE, CONNECT, OPTIONS 和 TRACE，最常用的请求方法是 GET 和 POST，GET 指的是从服务端获取资源，POST 是向指定资源提交数据进行请求处理。

9.2 HTTP 的状态码

在 HTTP 的协议中，客户端发送请求给服务端，服务端响应回复给客户端的同时，还会带上 HTTP 协议的状态码，具体状态码介绍如下。

1. 状态码之 200

在 HTTP 的协议中，状态码 200 是请求成功，也就是说客户端发送请求给服务端，服务端响应回复给客户端这个过程是成功的。例如查看博客园中我的粉丝，服务端返回的状态码如下。

General 信息：

```
Request URL: https://home.cnblogs.com/u/weke/followers/  
Request Method: GET  
Status Code: 200  
Remote Address: 116.62.82.159:443  
Referrer Policy: no-referrer-when-downgrade
```

2. 状态码之 201

状态码 201 指的是用于创建服务器对象的请求，它的短语是 **Created**。下面通过一个具体实例来说明，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from flask import Flask,jsonify,request,abort,url_for,make_response
from flask_httpauth import HTTPBasicAuth

app=Flask(__name__)

datas=[
    {
        'userid':1,
        'username':'李四',
        'phone':'13484545190',
    }
]

@app.route('/user/',methods=['GET'])
def get_user():
    return jsonify({'datas':datas})

@app.route('/user/',methods=['POST'])
def create_user():
    if not request.json or not 'username' in request.json:
        abort(400)

    data={
        'userid':datas[-1]['userid']+1,
        'username':request.json['username'],
        'phone':request.json.get('phone','13487904597')
    }
    datas.append(data)
    return jsonify({'data':data}),201

if __name__ == '__main__':
    app.run(debug=True)
```

下面我们创建用户。创建用户的请求方法是 POST，请求参数是 username 和 phone，PostMan 的内容如图 9-2-1 所示。

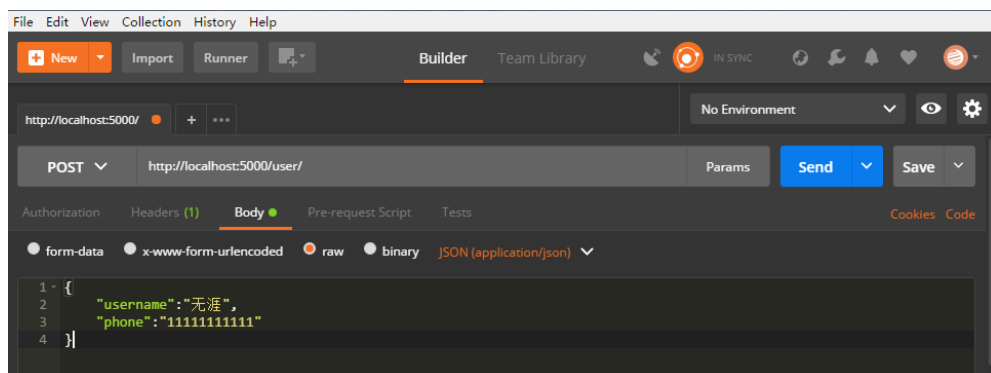


图 9-2-1

点击“Send”按钮后服务端的响应数据如图 9-2-2 所示。

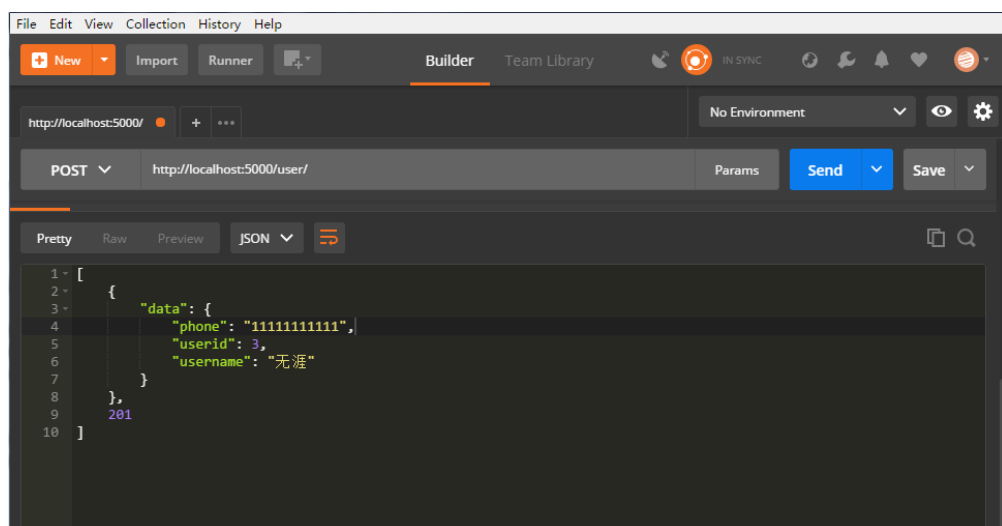


图 9-2-2

3. 状态码之 302

状态码 302 是临时重定向，例如，在博客园中，在未登录的情况下需要查看我的粉丝，就会把“我的粉丝”URL 重定向到博客园的登录页面。

General 的信息：

```
Request URL: https://home.cnblogs.com/u/weke/followers/
Request Method: GET
Status Code: 302
Remote Address: 116.62.82.159:443
Referrer Policy: no-referrer-when-downgrade
```

Response Headers 的信息:

```
content-length: 0
date: Sun, 08 Jul 2018 08:01:59 GMT
location:
http://home.cnblogs.com/account/signin?ReturnUrl=%2Fu%2Fweke%2Ffollowers%2F
server: Tengine
status: 302
```

Request Headers 的信息:

```
:authority: home.cnblogs.com
:method: GET
:path: /u/weke/followers/
:scheme: https
accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/a
png,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: zh-CN,zh;q=0.9
cookie: _ga=GA1.2.776381712.1530781235; _gid=GA1.2.1144888681.1531035066
dnt: 1
referer: https://www.cnblogs.com/weke/category/831885.html
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36
```

4. 状态码之 400

客户端向服务端发送请求，服务端返回 400 BAD REQUEST，指的是客户端请求的语法错误，服务端无法理解。还是以添加用户例，我们刻意把请求参数 username 写成 user，发送请求后，服务端返回的状态码是 400 BAD REQUEST，如图 9-2-3 所示。

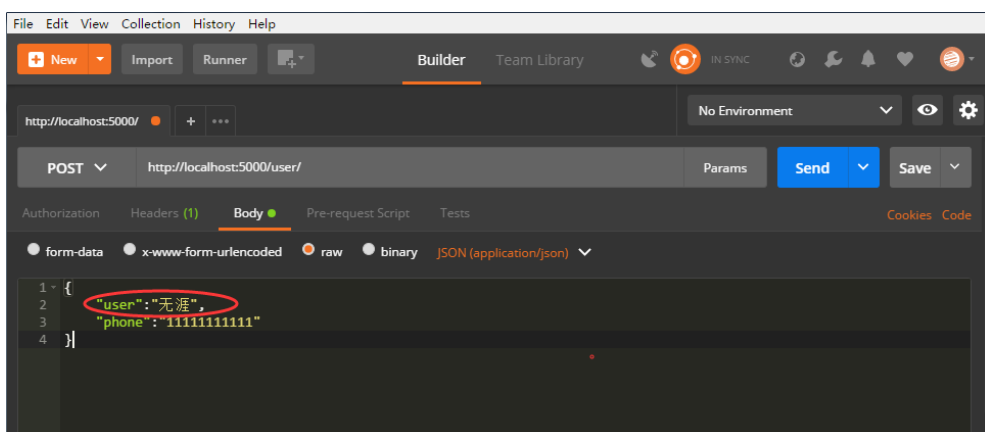


图 9-2-3

点击“Send”按钮发送请求后，服务端的响应回复内容如图 9-2-4 所示。

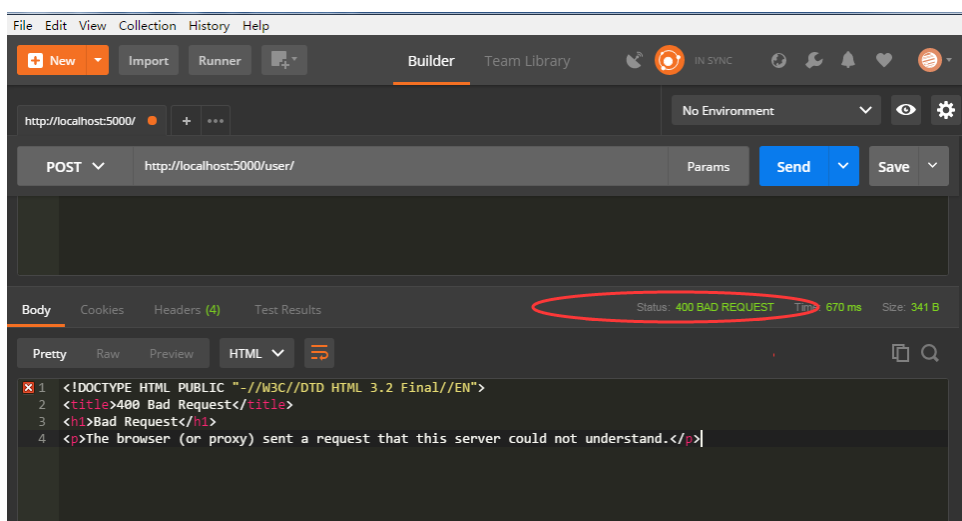


图 9-2-4

5. 状态码之 404

当客户端向服务端发送请求，但请求的 URL 在服务器里不存在，此时会返回状态码 404 NOT FOUND。在实例中，获取用户信息的 URL 是 `http://localhost:5000/user/`，而刻意写成 `http://localhost:5000/username/`，发送请求，服务端就会返回状态码为 404，如图 9-2-5 所示。

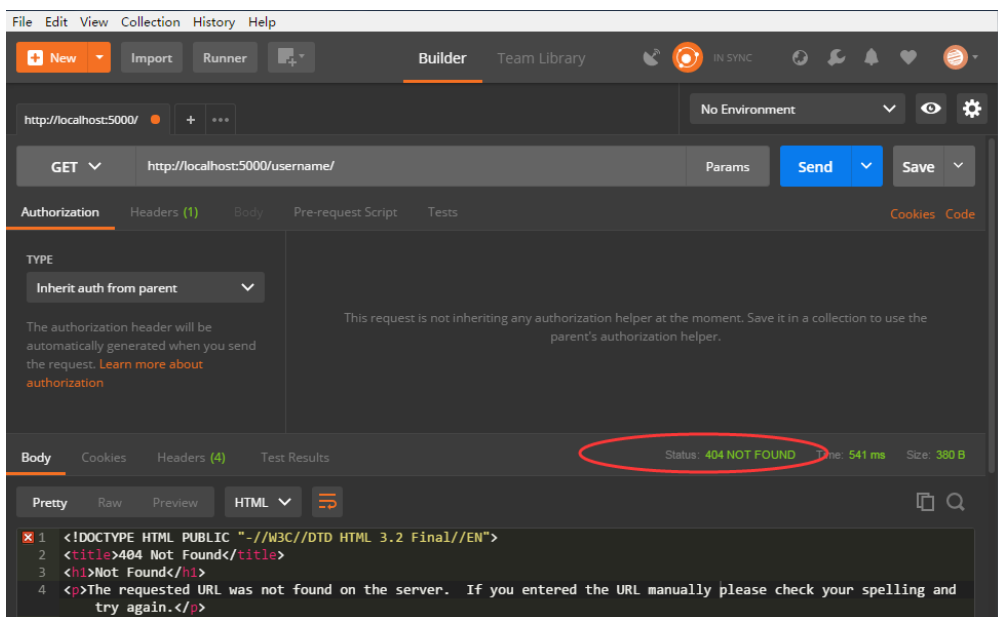


图 9-2-5

6. 状态码之 401

状态码 401 指的是被请求的页面需要账号和密码。对以上实例进行完善后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from flask import Flask,jsonify,request,abort,url_for,make_response
from flask_httpauth import HTTPBasicAuth

app=Flask(__name__)

'''增加 HTTPBasicAuth 的权限验证机制'''
auth=HTTPBasicAuth()

@auth.get_password
def get_password(username):
    if username=='wuya':
        return 'admin'
    return None
```

```

@auth.error_handler
def unauthorized():
    return make_response(jsonify({'error': 'Unauthorized access'}), 401)

datas=[
    {
        'userid':1,
        'username':'李四',
        'phone':'13484545190',
    }
]

@app.route('/user/', methods=['GET'])
@auth.login_required
def get_user():
    return jsonify({'datas':datas})

@app.route('/user/', methods=['POST'])
@auth.login_required
def create_user():
    if not request.json or not 'username' in request.json:
        abort(400)

    data={
        'userid':datas[-1]['userid']+1,
        'username':request.json['username'],
        'phone':request.json.get('phone', '13487904597')
    }
    datas.append(data)
    return jsonify({'data':data}, 201)

if __name__ == '__main__':
    app.run(debug=True)

```

注释：在以上代码中，增加了基本权限的认证，也就是说访问的时候，如果没有填写用户名和密码，就会返回 401 的状态码。

运行以上代码后，在 PostMan 中访问 <http://localhost:5000/user/>，服务端就会返回 401 的错误，如图 9-2-6 所示。

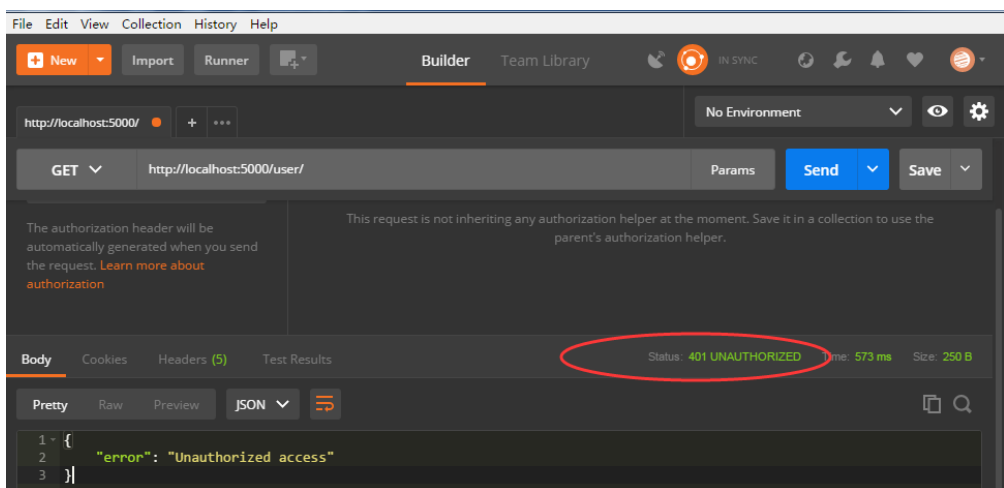


图 9-2-6

如果在浏览器访问 `http://localhost:5000/user/`，就会弹出要求输入账号和密码的输入框，如图 9-2-7 所示。

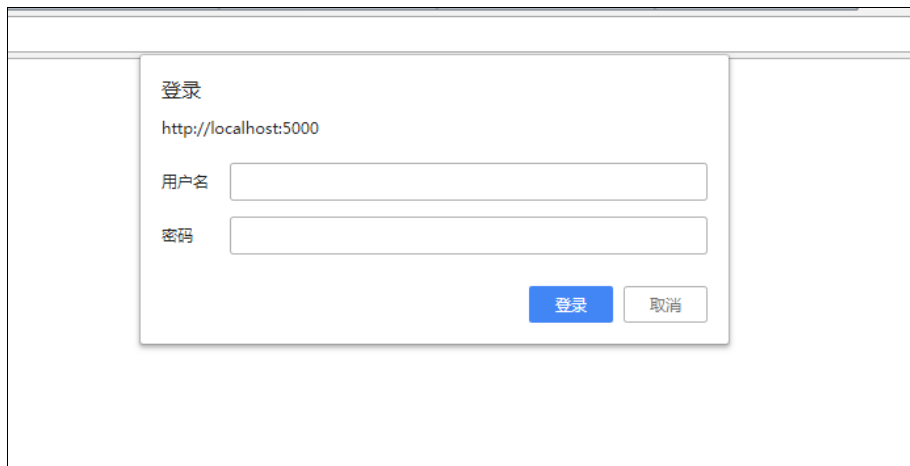


图 9-2-7

打开 Chrome 浏览器调试功能的 Network，在弹出框中点击“取消”按钮后，General 和 Response Headers 的信息分别如下。

General 的信息：

```
Request URL: http://localhost:5000/user/
Request Method: GET
Status Code: 401 UNAUTHORIZED
```



```
Remote Address: 127.0.0.1:5000
Referrer Policy: no-referrer-when-downgrade
```

Response Headers 的信息:

```
Content-Length: 37
Content-Type: application/json
Date: Sun, 08 Jul 2018 10:17:21 GMT
Server: Werkzeug/0.12.2 Python/3.6.1
WWW-Authenticate: Basic realm="Authentication Required"
```

依据以上信息, 找到响应头中 **WWW-Authenticate**, 它指向的是客户端应该在请求头中提供什么类型的授权信息, 也就是说提供 **Basic** 基本认证的授权信息。再次请求, 输入用户名和密码, 请求成功后, 在请求头中会带有 **Authorization** 对应的内容, 如图 9-2-8 所示。

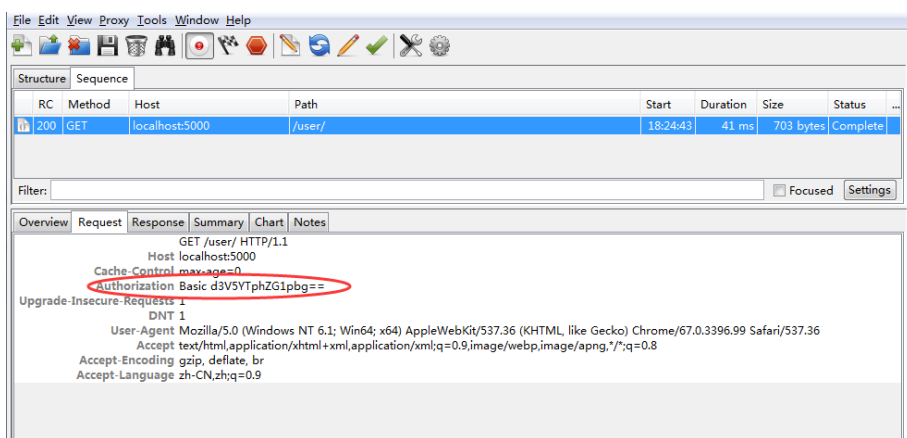


图 9-2-8

Charles 工具中的 Authenticaion 部分如图 9-2-9 所示。

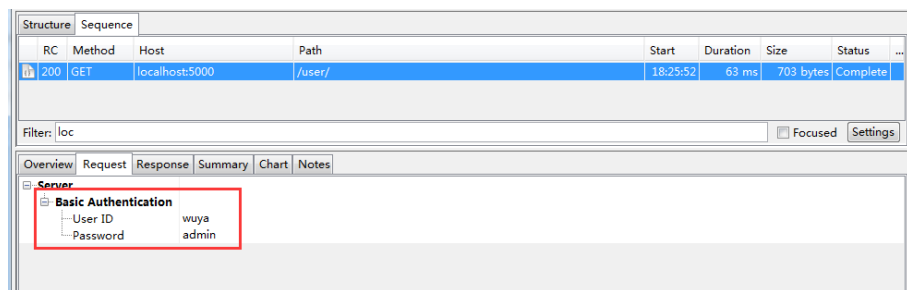


图 9-2-9

7. 状态码之 403

状态码 403 指的是服务器理解客户端的请求，但是拒绝执行。通俗地说就是客户端去请求某一资源，由于权限等原因，服务器拒绝客户端的请求。依然以以上代码为例，在函数 `unauthorized` 中把 401 修改为 403，修改后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from flask import Flask,jsonify,request,abort,url_for,make_response
from flask_httpauth import HTTPBasicAuth

app=Flask(__name__)

'''增加 HTTPBasicAuth 的权限验证机制'''
auth=HTTPBasicAuth()

@auth.get_password
def get_password(username):
    if username=='wuya':
        return 'admin'
    return None

@auth.error_handler
def unauthorized():
    return make_response(jsonify({'error':'Unauthorized access'}),403)

datas=[
    {
        'userid':1,
        'username':'李四',
        'phone':'13484545190',
    }
]

@app.route('/user/',methods=['GET'])
@auth.login_required
def get_user():
    return jsonify({'datas':datas})

@app.route('/user/',methods=['POST'])
@auth.login_required
def create_user():
```

```

if not request.json or not 'username' in request.json:
    abort(400)

data={
    'userid':datas[-1]['userid']+1,
    'username':request.json['username'],
    'phone':request.json.get('phone','13487904597')
}
datas.append(data)
return jsonify({'data':data},201)

if __name__ == '__main__':
    app.run(debug=True)

```

再次运行以上代码，在浏览器中输入 `http://localhost:5000/user/`，按下回车键，打开浏览器的调试功能，在 **Network** 下，可以看到服务器的状态码是 403，如图 9-2-10 所示。

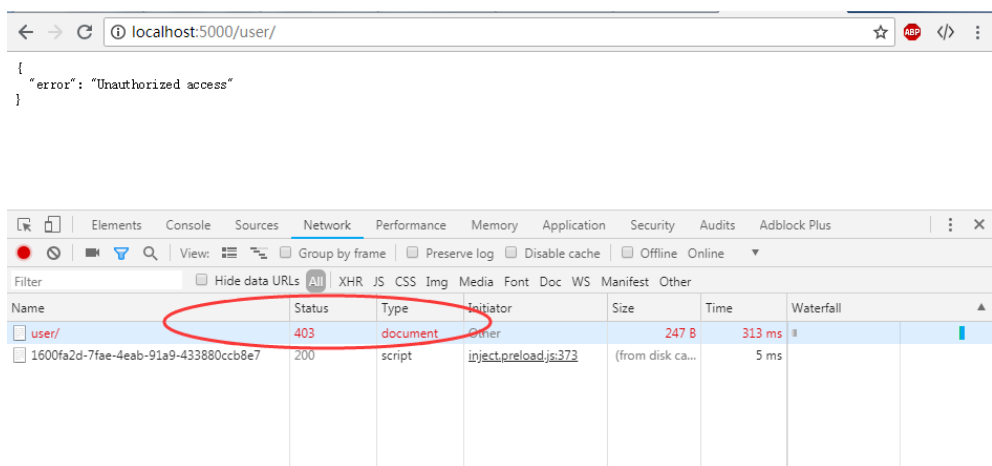


图 9-2-10

点击查看详细信息如下。

General 的信息：

```

Request URL: http://localhost:5000/user/
Request Method: GET
Status Code: 403 FORBIDDEN
Remote Address: 127.0.0.1:5000
Referrer Policy: no-referrer-when-downgrade

```

Response Headers 的信息:

```
Content-Length: 37
Content-Type: application/json
Date: Mon, 09 Jul 2018 14:01:03 GMT
Server: Werkzeug/0.12.2 Python/3.6.1
WWW-Authenticate: Basic realm="Authentication Required"
```

8. 状态码之 500

状态码 500 表示客户端发送请求到服务端后, 服务端内部发生不可预知的错误导致客户端的请求未完成。对之前的代码进行修改, 在 GET 请求中刻意把返回的数据写错, 修改后的代码如下:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

from flask import Flask,jsonify,request,abort,url_for,make_response
from flask_httpauth import HTTPBasicAuth

app=Flask(__name__)

'''增加 HTTPBasicAuth 的权限验证机制'''
auth=HTTPBasicAuth()

@auth.get_password
def get_password(username):
    if username=='wuya':
        return 'admin'
    return None

@auth.error_handler
def unauthorized():
    return make_response(jsonify({'error':'Unauthorized access'}),401)

datas=[
    {
        'userid':1,
        'username':'李四',
        'phone':'13484545190',
    }
]
```

```
@app.route('/user/',methods=['GET'])
@auth.login_required
def get_user():
    return ({'datas':datas})

if __name__ == '__main__':
    app.run(debug=True)
```

在浏览器中访问 <http://localhost:5000/user/>地址，输入用户名和密码后，按下回车键，服务端返回错误信息，在浏览器的 Network 中 General 显示的信息为：

```
Request URL: http://localhost:5000/user/
Request Method: GET
Status Code: 500 INTERNAL SERVER ERROR
Remote Address: 127.0.0.1:5000
Referrer Policy: no-referrer-when-downgrade
```

在接口测试中，当客户端发送请求后，如果服务端返回的状态码是 500 INTERNAT SERVER ERROR，一般是由于服务端发生错误导致的，建议把这些信息反馈给后台的程序员去定位具体的错误原因。

9.3 Cookie 的请求流程

由于 HTTP 协议是一个无状态的协议，所以也就导致了 Cookie 技术的发展。Cookie 存储在客户端，它的工作是进行用户识别和状态管理，也就是说网站为了管理用户的状态会通过 Web 浏览器，把一些数据临时写入用户的计算机内，用户再次访问 Web 站点的时候，可通过通信方式取回之前存放的 Cookie，在调用 Cookie 时候，同时会检验 Cookie 的时效。例如，在百度搜索中，搜索设置为显示搜索历史记录，那么搜索的时候会显示上次搜索的关键字。

这里结合 Python 的 Web 开发框架 Django 来看一个 Cookie 的实例，实例是登录系统成功后，显示登录成功后的用户账号（昵称），Views 的代码如下：

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.shortcuts import render,redirect,render_to_response
from django.http import HttpResponseRedirect
```

```
# Create your views here.

def index(request):
    username = request.COOKIE.get('user', None)
    return render(request, 'blog/index.html', locals())

def login(request):
    if request.method == 'GET':
        return render(request, 'blog/login.html', locals())
    if request.method == 'POST':
        username = request.POST.get('username', None)
        password = request.POST.get('password', None)
        if username == 'wuya' and password == 'admin':
            r = redirect('blog:index')
            r.set_cookie('user', username)
            return r
    return render(request, 'blog/login.html', locals())
```

注解：在以上代码中，在函数 `login` 中的用户名和密码是 `wuya` 和 `admin`，登录系统成功后跳转到首页，并且以 `Cookie` 的形式记录下登录系统的账号。

启动 Django 的服务后，在浏览器访问 `http://localhost:8000/blog/login`，输入用户名 `wuya` 和密码 `admin` 后，点击“登录”按钮跳转到 `http://localhost:8000/blog/` 的页面，在登录发送请求后，服务端会通过 `Set-Cookie` 记录下登录成功后的 `Cookie`，也就是用户的登录账号和其他的信息，见 `login` 登录请求后 `Response Headers` 的信息：

```
Content-Length: 0
Content-Type: text/html; charset=utf-8
Date: Sun, 29 Jul 2018 15:02:32 GMT
Location: /blog/
Server: WSGIServer/0.1 Python/2.7.12
Set-Cookie: user=wuya; expires=Sun, 29-Jul-2018 16:02:32 GMT; Max-Age=3600; Path=/
X-Frame-Options: SAMEORIGIN
```

注解：在 `Set Cookie` 中，记录了登录系统的账号 `wuya`，`expires` 表示 `Cookie` 的过期时间，`Path` 是 `Cookie` 的路径。

登录成功后，再次请求 `index` 的首页，客户端在发送 `index` 请求的时候会在请求头中带上 `Cookie` 的值，表示用户已登录成功，`index` 请求的 `Request Headers`

的信息为:

```
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/a
png,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Cookie: user=wuya
DNT: 1
Host: localhost:8000
Referer: http://localhost:8000/blog/login/
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/68.0.3440.75 Safari/537.36
```

注解: 在请求头中可以看到 Cookie 对应的值是 user=wuya。

依据以上的请求, 总结出 Cookie 的请求流程如图 9-3-1 所示。

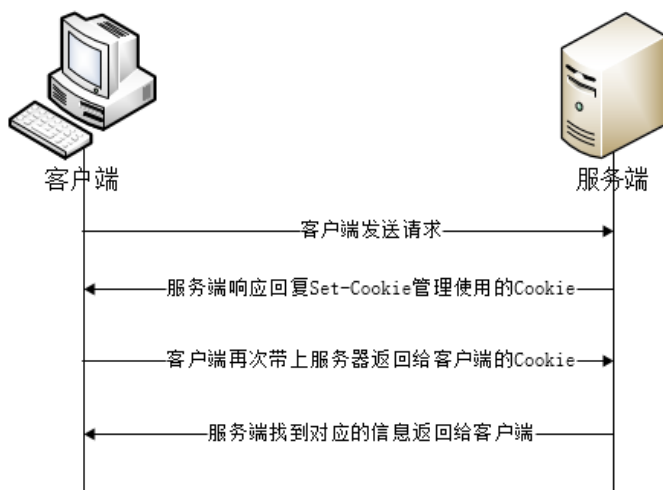


图 9-3-1

依据图 9-3-1 可以得到 Cookie 的请求流程为:

- (1) 客户端(<http://localhost:8000/blog/login/>)发出请求到服务端;
- (2) 服务端得到客户端的请求后, 在响应头中 Set-Cookie 记录下 Cookie 并反馈给客户端, 客户端会将收到的 Cookie 信息本地化, 即存储起来;

(3) 客户端再次发送请求的时候(<http://localhost:8000/blog/index/>), 在请求头添加 **Cookie** 后发送给服务端;

(4) 服务端将接收到的 **Cookie** 信息与存储在本地的 **Cookie** 进行检查校验, 如果一致, 就显示登录系统成功后的信息; 如果不一致, 直接跳转到登录的页面。

9.4 Session 的请求流程

Cookie 存储在客户端, 而对于一些敏感的信息来说, 存储在本地显然是不安全的, 例如, 登录系统的账号和密码, 被别人获取后会对个人的隐私造成影响。**Session** 则将信息存储在服务端。使用 **Session** 来保持会话, 也就是说客户端发送请求后, 服务端在接收到客户端的信息后, 把 **Session** 的信息存储在服务端的数据库中并且记录下来。还是以以上代码为例, 把实现的方式修改为 **Session**, 修改后的代码如下:

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.shortcuts import render, redirect, render_to_response
from django.http import HttpResponseRedirect

# Create your views here.

def index(request):
    username = request.session.get('user', None)
    return render(request, 'blog/index.html', locals())

def article(request, year, month):
    return HttpResponseRedirect('2004')

def login(request):
    if request.method == 'GET':
        return render(request, 'blog/login.html', locals())
    if request.method == 'POST':
        username = request.POST.get('username', None)
        password = request.POST.get('password', None)
        if username == 'wuya' and password == 'admin':
            r = redirect('blog:index')
            request.session['user'] = username
            return r
    return render(request, 'blog/login.html', locals())
```


再次在浏览器中访问 `http://localhost:8000/blog/login`，打开浏览器的调试模式到 Network，输入用户名和密码，点击“登录”按钮登录到系统。`http://localhost:8000/blog/login` 的 Response Headers 信息如下：

```
Content-Length: 0
Content-Type: text/html; charset=utf-8
Date: Mon, 30 Jul 2018 14:12:51 GMT
Location: /blog/
Server: WSGIServer/0.1 Python/2.7.12
Set-Cookie: sessionid=o3imtvyrpnfkavmyjo4licw7r98pu6l2; expires=Mon, 13-Aug-2018 14:12:51 GMT; httponly; Max-Age=1209600; Path=/
Vary: Cookie
X-Frame-Options: SAMEORIGIN
```

在以上的 Set-Cookie 服务端会记录客户端登录成功后的 SessionID 并且存储在服务端的数据库中，数据库存储的 SessionID 如图 9-4-1 所示。

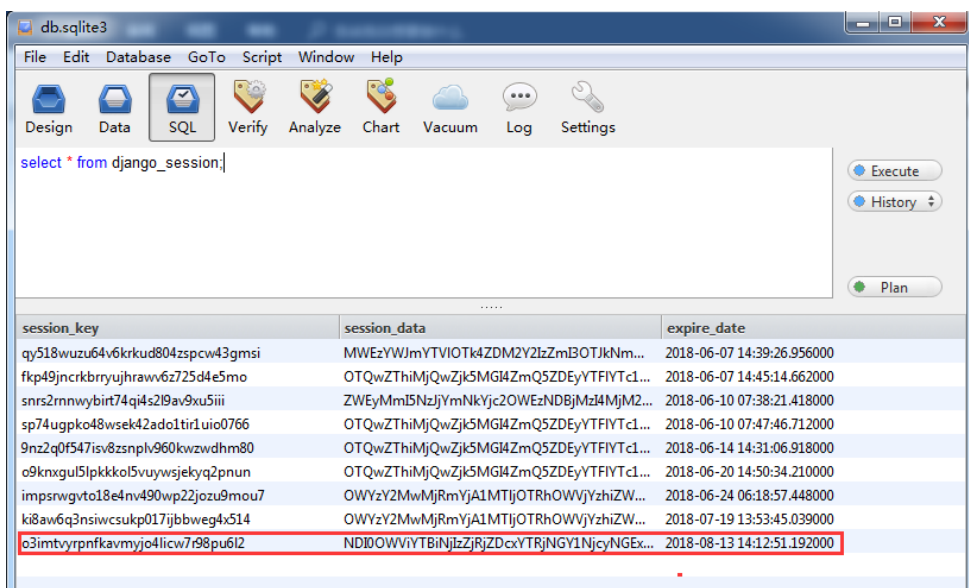


图 9-4-1

注解：在图 9-4-1 中可以看到，记录的 session_key 和服务端响应头的 Set-cookie 中的 SessionID 一致。

登录系统成功后，客户端请求头的信息如下：

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cache-Control: max-age=0
Connection: keep-alive
Cookie: sessionId=o3imtvyrpnfkavmyjo4licw7r98pu6l2
DNT: 1
Host: localhost:8000
Referer: http://localhost:8000/blog/login/
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.75 Safari/537.36
```

客户端请求头的 Cookie 中的 SessionID 与存储在服务端的 SessionID 信息一致，因此 Session 的请求流程为：

- (1) 客户端中输入用户名和密码，点击“登录”按钮发送请求到服务端；
- (2) 登录成功后，服务端把 SessionID 存储在数据库中并且在响应头 Set-Cookie 中记录下 SessionID 反馈给客户端；
- (3) 客户端再次发送请求的时候，在请求头 Cookie 带上 SessionID 信息；
- (4) 服务端接收到客户端发送的 SessionID，会与存储在服务端数据库中的 SessionID 做校验检查是否存在，如果存在则显示用户的信息；如果不存在就跳转到登录的页面。

9.5 Token 的请求流程

在接口测试中，还有一种产品处理的方式是 Token。那么，什么是 Token 呢？Token 可以理解为令牌，原理上是通过 Session 实现的。成功登录一个系统后，服务端会生成一个 32 位随机的字符串（也就是 Token），Token 存放在 Session 中并且会返回给客户端，当客户端再次发出请求时，需要把 Token 带上并且该 Token 的值与登录成功后返回的 Token 值一致，否则服务端会返回无效的请求，如图 9-5-1 所示。

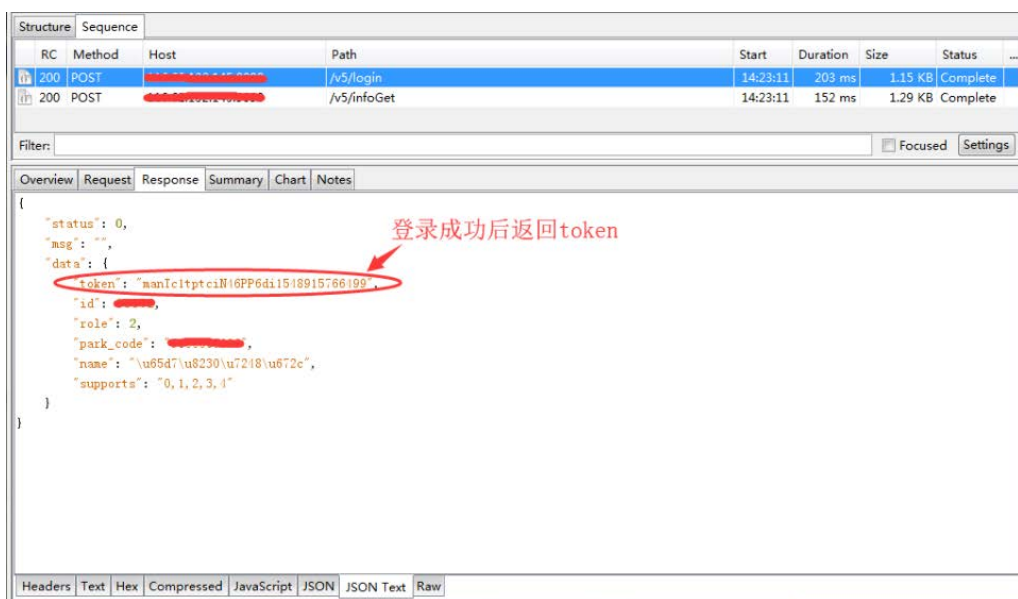


图 9-5-1

登录成功后的 token 值如图 9-5-2 所示。

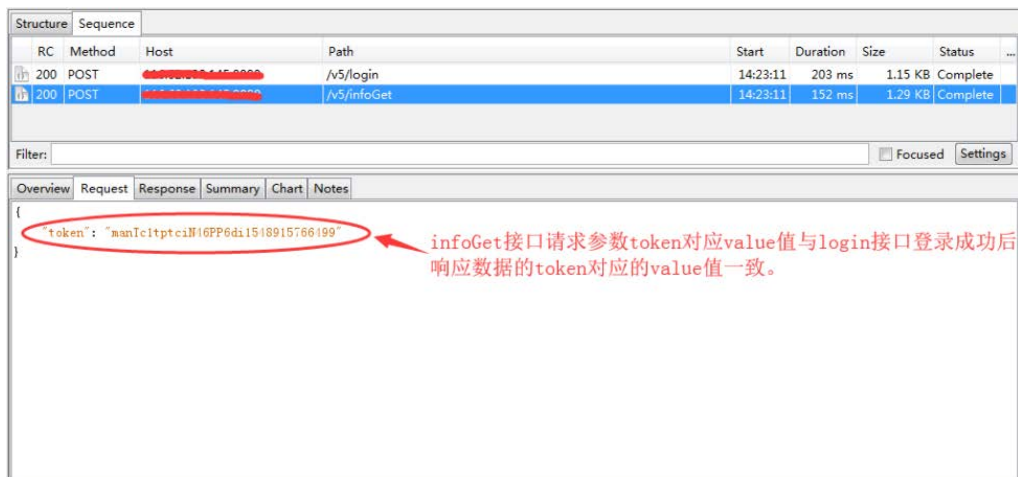


图 9-5-2

依据图 9-5-2 信息，可以总结出 Token 的请求流程为：

- (1) 客户端使用账号和密码登录到系统；
- (2) 服务端接收到客户端请求的参数后，验证用户名和密码是否正确；

(3) 验证用户名和密码成功后，服务端会签发一个 Token 的值，并且把 Token 返回给客户端；

(4) 客户端获取到 Token 后，客户端会把 Token 存储起来；

(5) 客户端再次向服务端发送请求的时候，需要带着服务端签发的 Token，服务端接收到客户端发送的 Token 会进行校验，校验通过后服务端会返回客户端请求的响应数据。

第 10 章 序列化与反序列化

在 Python 中，序列化指的是把 Python 的对象编码转换为 JSON 格式的字符串；反序列化则相反，是把 JSON 格式字符串解码为 Python 数据对象。在 Python 标准库中，专门提供了 JSON 库来处理这个问题。

10.1 JSON 库的应用

在 Python 中，把内置数据结构如元组、字典、列表进行序列化处理后，类型为 str，而经过反序列化处理后，数据类型依然是列表和字典（元组经过序列化和反序列化后数据类型是字典，不再是元组）。在 JSON 库中，序列化和反序列化的处理分为两部分，一部分是对具体列表数据的处理，另外一部分是对文件内容的处理。下面通过具体的实例代码，来介绍其实现的过程，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import json

list1=[1,2,3,4,5]
print ('\n 对列表进行序列化和反序列化的处理：')

print ('列表未序列化之前的数据类型:',type(list1))
#对列表 list1 进行序列化处理
list_str=json.dumps(list1)
print ('列表 list1 序列化后的内容:{0}和类型{1}'.format(list_str,type(list_str)))

#对字符串 list_str 进行反序列化处理
str_list=json.loads(list_str)
print ('字符串 list_str 反序列化后的内容:{0}和类
```

```

型:{1}'.format(str_list,type(str_list)))

tuple1=('name','wuya','age')
print ('\n 对元组进行序列化和反序列化的处理: ')

print ('元组未序列化之前的数据类型:',type(tuple1))
#对元组 tuple1 进行序列化处理
tuple_str=json.dumps(tuple1)
print ('元组 tuple1 序列化后的内容:{0}和类型
{1}'.format(tuple_str,type(tuple_str)))

#对字符串 tuple_str 进行反序列化处理
str_tuple=json.loads(tuple_str)
print ('字符串 tuple_str 反序列化后的内容:{0}和类
型:{1}'.format(str_tuple,type(str_tuple)))

dict1={'name':'wuya','address':'wuya','age':'20'}
print ('\n 对字典进行序列化与反序列化的处理: ')

print ('字典未序列化之前的数据类型:',type(dict1))
#对字典 dict1 进行序列化处理
dict_str=json.dumps(dict1)
print ('字典 dict_str 序列化后的内容:{0}和类型
{1}'.format(dict_str,type(dict_str)))

#对字符串 dict_str 进行反序列化处理
str_dict=json.loads(dict_str)
print ('字符串 dict_str 反序列化后的内容:{0}和类
型:{1}'.format(str_dict,type(str_dict)))

```

在接口自动化测试中，客户端发送请求到服务端，服务端响应回复数据给客户端，客户端拿到响应数据后就可以把这些数据存储在文件中。在对文件的处理中，序列化的过程实际上是把数据存储在文件的过程，反序列化的过程是读取文件里面的内容。在下面的实例中我们请求一个接口，然后把服务端返回的响应数据存储在文件 json.md 中，经过反序列化处理后，取出里面具体的数据值，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

```

```
'''
请求接口 http://118.***.***.145:9999/v5/login,把服务端返回的响应
数据存储在文件 json.md 中,然后对文件 json.md 反序列化处理获取里面
具体的值。
'''

import requests
import json

def login():
    headers={
        'Content-Type':'application/json;charset=UTF-8',
        'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'
    }
    data={"username":"","password":""}
    r=requests.post(
        url='http://118.***.***.145:9999/v5/login',
        json=data,
        headers=headers)
    #把服务端返回的响应数据存储在文件 json.md 中
    json.dump(r.json(),open('json.md','w'))

login()

#对文件 json.md 进行反序列化处理,取值 msg 对应的 value 具体值
dict1=json.load(open('json.md','r'))
print ('文件 json.md 反序列化后的内容{0}和类型{1}'.format(dict1,type(dict1)))
print ('msg 对应的 value 为: {0}'.format(dict1['msg']))
```

10.2 JSON 库的实例实战

在找工作时,经常会使用到拉勾网这样的互联网平台,这里结合 Requests 库来进行一个接口测试:在拉勾网平台搜索“自动化测试工程师”,然后把获取的数据进行反序列化的处理,从而对招聘单位的最高薪资和最低薪资进行数据分析。首先在拉勾网平台搜索关键字,然后查看返回列表的接口请求数据,如图 10-1-1 所示。

▼ Form Data view source view URL encoded

```
first: true
pn: 1
kd: 自动化测试工程师
```

图 10-1-1

在拉勾网搜索“自动化测试工程师”的接口测试，实现的代码如下：

```
import requests

def headers():
    headers={
        'Content-Type':'application/x-www-form-urlencoded; charset=UTF-8',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110
Safari/537.36',

        'Referer':'https://www.lagou.com/jobs/list_%E8%87%AA%E5%8A%A8%E5%8C%96%E
6%B5%8B%E8%AF%95?labelWords=&fromSearch=true&suginput=',
        'Cookie':'_ga=GA1.2.1408985754.1541511212;
user_trace_token=20181106213332-8de613ff-e1c8-11e8-86e9-5254005c3644;
LGUID=20181106213332-8de616ff-e1c8-11e8-86e9-5254005c3644;
index_location_city=%E5%85%A8%E5%9B%BD;
JSESSIONID=ABAAABAAAGGABCBF9FAD2B1E4E827B38E71F34EA637EC24; _gat=1;
_gid=GA1.2.1661630433.1543413992; LGSID=20181128220628-cc8993f9-f316-
11e8-8c3c-5254005c3644; PRE_UTM=; PRE_HOST=; PRE_SITE=;
PRE_LAND=https%3A%2F%2Fwww.lagou.com%2F;
Hm_lvt_4233e74dff0ae5bd0a3d81c6ccf756e6=1542269661,1542721737,1543149533
,1543413992; SEARCH_ID=b9bc6a48f9ca40ea863a8ac7dac68b34;
LGRID=20181128220635-d0f99943-f316-11e8-83f9-525400f775ce;
Hm_lpv_4233e74dff0ae5bd0a3d81c6ccf756e6=1543414000; TG=TRACK-
CODE=search_code'}
    return headers

def laGou():
    r=requests.post(

url='https://www.lagou.com/jobs/positionAjax.json?needAdditionalResult=false',
        data={'first':False,'pn':1,'kd':'自动化测试工程师'},
        headers=headers())
    print(r.text)
```

接下来要实现的是获取每个招聘公司的公司名称和公司薪资。执行 laGou 函数后，虽然返回了招聘的整个信息，但是它的类型是字符串，无法获取到具体的每个公司的名称，和招聘职位的薪资。在这里使用 JSON 的库，进行序列化的处理，把服务端返回的数据写到文件中，然后再对文件中的数据进行进一步的分析。再次修改 laGou 函数，实现把服务端返回的响应数据写到 lagou.json 文件中，修

改后的代码如下：

```
def laGou():
    r=requests.post(

url='https://www.lagou.com/jobs/positionAjax.json?needAdditionalResult=false',
    data={'first':False,'pn':1,'kd':'自动化测试工程师'},
    headers=headers())
    json.dump(r.json(),open('lagou.json','w'))
```

执行 laGou 函数后，把获取到的数据存储到 lagou.json 文件中，接下来读取该文件里面的数据，获取到招聘职位的公司名称、最高薪资，然后生成可视化的数据分析的一个页面。读取 lagou.json 文件里面的内容，需要进行反序列化的处理，会使用到 JSON 库中的 load 方法。编写函数 dataAndlysis，代码如下：

```
def dataAnalysis():
    positions=[]
    data=json.load(open('lagou.json','r'))
    #获取招聘职位公司名称
    for i in range(15):
        company =
data['content']['positionResult']['result'][i]['companyFullName']
        salary = data['content']['positionResult']['result'][i]['salary']
        positions.append({
            'company':company,
            'salary':salary
        })
```

接下来对公司名称和薪资进行处理，把公司名称单独放到一个列表中，把薪资也单独放到一个列表中，然后获取薪资范围的最高薪资，见修改后的 dataAnalysis 函数：

```
def dataAnalysis():
    positions=[]
    data=json.load(open('lagou.json','r'))
    #获取招聘职位公司名称
    for i in range(15):
        company =
data['content']['positionResult']['result'][i]['companyFullName']
        salary = data['content']['positionResult']['result'][i]['salary']
        positions.append({
            'company':company,
```

```

        'salary':salary
    })
    company=list(map(lambda x:x['company'],positions))
    salary=list(map(lambda x:x['salary'],positions))
    salaryMax=[]
    # 获取最高薪资
    for item in salary:
        salaryMax.append(int(str(item.split('-')[1]).split('k')[0]))

```

注释：在如上代码中，公司名称和薪资都在列表 `positions` 中，然后从该列表中单独得到公司名称和薪资，这里使用了内部函数 `map` 和 `lambda` 匿名函数。由于薪资是一个范围，所以需要对获取到的薪资进行特别的处理。获取到最高薪资并且取消薪资后面的字母 `k`，然后把薪资从字符串的类型转为 `int` 类型的数据，并且添加到列表 `salaryMax` 中。

接下来使用 `Pycharts` 库实现可视化，修改后的 `dataAnalysis` 函数代码如下：

```

from pyecharts import Bar

def dataAnalysis():
    positions=[]
    data=json.load(open('lagou.json','r'))
    #获取招聘职位公司名称
    for i in range(15):
        company =
data['content']['positionResult']['result'][i]['companyFullName']
        salary = data['content']['positionResult']['result'][i]['salary']
        positions.append({
            'company':company,
            'salary':salary
        })
    company=list(map(lambda x:x['company'],positions))
    salary=list(map(lambda x:x['salary'],positions))
    salaryMax=[]
    # 获取最高薪资
    for item in salary:
        salaryMax.append(int(str(item.split('-')[1]).split('k')[0]))
    bar=Bar('招聘平台薪资数据分析')
    bar.use_theme('dark')
    bar.add('薪资分析',company,salaryMax,is_legend_show=True)
    bar.render('lagou.html')

```

执行 `dataAnalysis` 函数后，会在当前目录下生成 `lagou.html` 文件，打开该 HTML 文件就会显示出每个公司招聘的最高薪资，如图 10-1-2 所示。

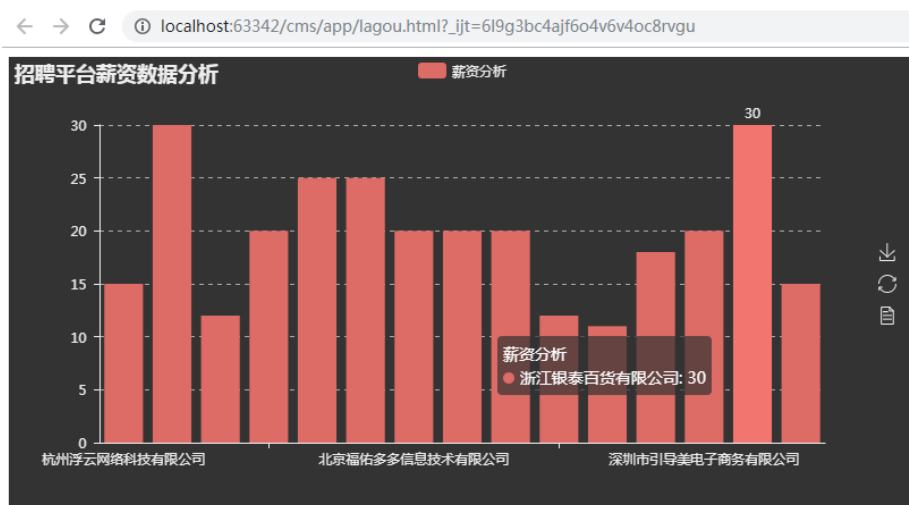


图 10-1-2

第 11 章 PostMan 的应用

11.1 PostMan 简述

PostMan 是一款用于发送 HTTP 请求的 Chrome 插件，功能超级强大，使用非常广泛，在功能测试工作中，经常会使用到 PostMan 来做接口测试，如查看一个接口中服务端返回的响应数据。PostMan 为本地应用程序读者可以人 <https://www.getpostman.com/> 网站中下载 PostMan 的安装包，下载后按提示进行安装。安装成功后，启动 PostMan，它的默认界面如图 11-1-1 所示。

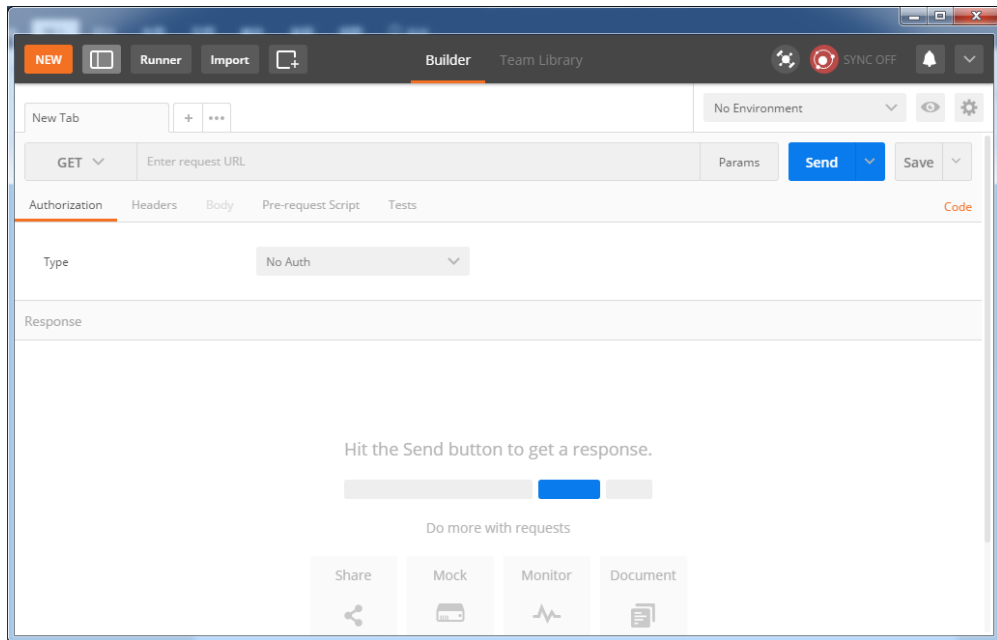


图 11-1-1

11.2 PostMan 实战

1. 发送请求

下面来看 PostMan 的第一个请求。在输入框中输入 `postman-echo.com/get` 后，点击“Send”按钮，在 Response 区域就可以看到客户端发送请求后，服务端返回的响应数据，它的数据格式是基于 JSON 的字符串，同时显示该 HTTP 请求返回的 HTTP 协议状态码、时间和大小，如图 11-2-1 所示。

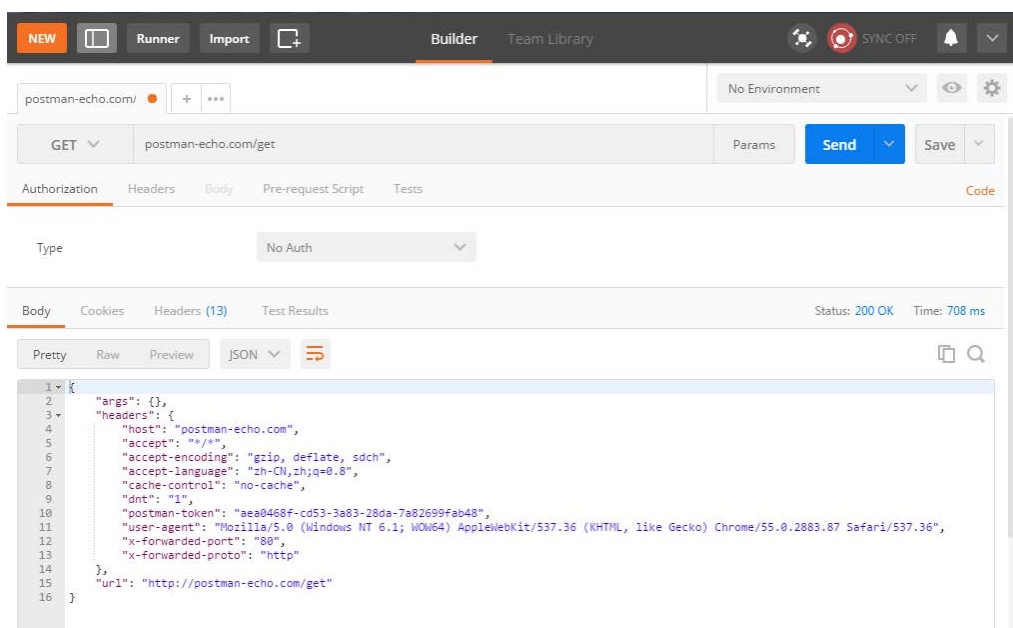


图 11-2-1

这样一个请求它是如何发生的呢？或者说 PostMan 是如何工作的呢？PostMan 工作流程如图 11-2-2 所示。

依据图 11-2-2 所示，它工作的方式为：

- (1) 输入请求地址的详细信息后，点击“Send”（发送）按钮；
- (2) 该请求由 API 服务器接收，不管请求成功还是失败，服务器都会返回一个响应报文；
- (3) 响应报文由 PostMan 接收后，会以可视化的方式显示出来。

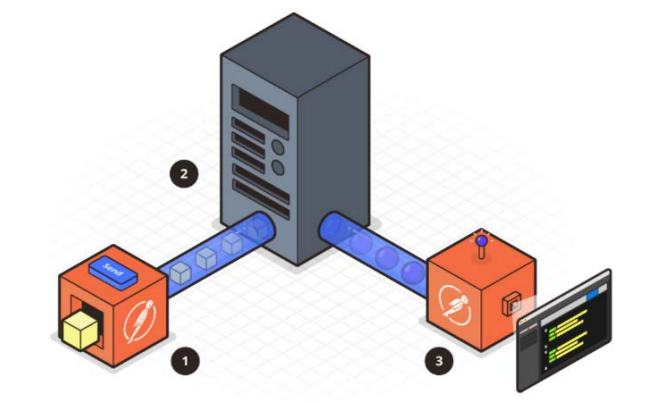


图 11-2-2

2. WebServices 的测试

在接口测试中，经常会遇到 **WebServices** 接口，下面通过一个具体实例进行详细介绍。通过电话号码查询该号码所属地，客户端请求信息和服务端返回的响应信息如下：

```
POST /WebServices/MobileCodeWS.asmx HTTP/1.1
Host: ws.webxml.com.cn
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://WebXml.com.cn/getMobileCodeInfo"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfo xmlns="http://WebXml.com.cn/">
      <mobileCode>string</mobileCode>
      <userID>string</userID>
    </getMobileCodeInfo>
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfoResponse xmlns="http://WebXml.com.cn/">
      <getMobileCodeInfoResult>string</getMobileCodeInfoResult>
    </getMobileCodeInfoResponse>
  </soap:Body>
</soap:Envelope>
```

从以上信息中可以得到，请求地址是 `http://ws.webxml.com.cn/WebServices/MobileCodeWS.asmx`；请求方法是 POST；请求参数是 `mobileCode` 和 `userID`，`userID` 参数可以为空；Content-Type 是 `text/xml; charset=utf-8`。

下面通过 PostMan 实现以上的接口请求过程。在 PostMan 中添加 Headers，填写请求地址和选择请求的方法，如图 11-2-3 所示。

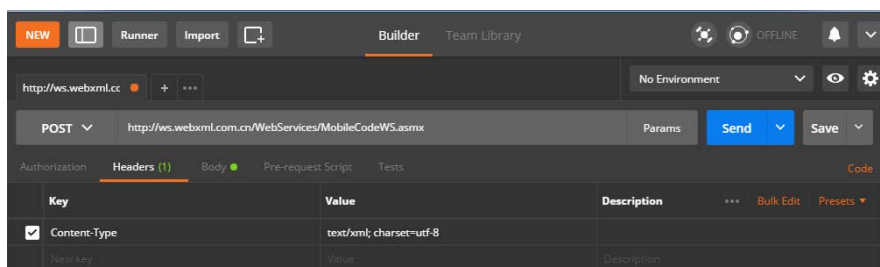


图 11-2-3

请求的参数如图 11-2-4 所示。

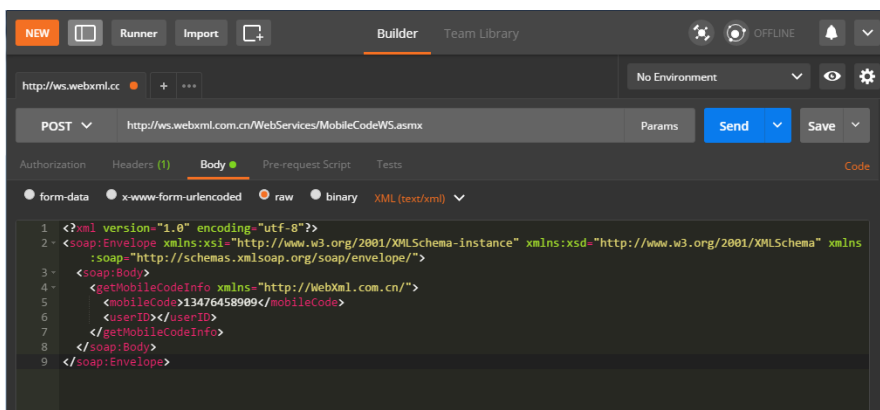


图 11-2-4

点击“Send”（发送）按钮后，服务端返回的响应数据如图 11-2-5 所示。

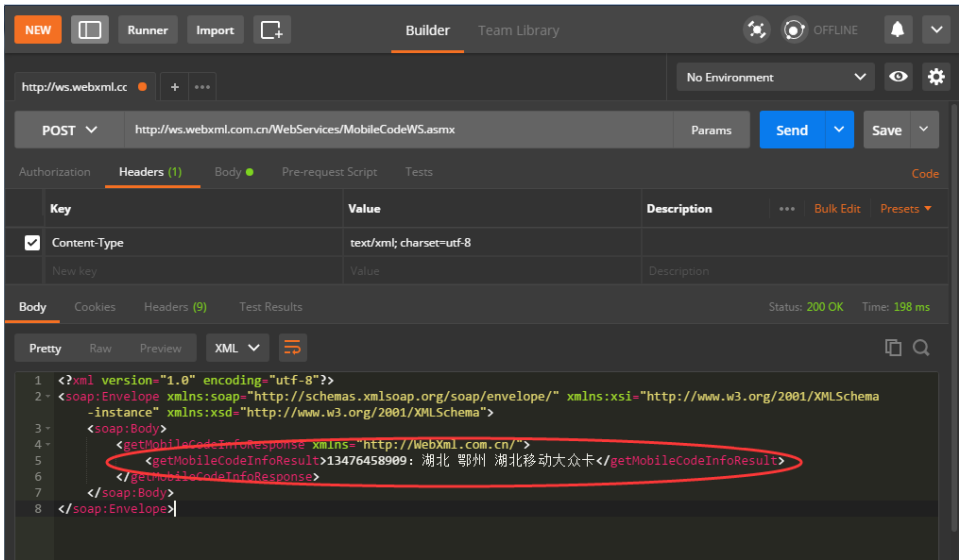


图 11-2-5

3. HTTP 的测试

在接口测试中应用广泛的，目前依然是基于 HTTP 协议的接口测试，如对一个程序发送请求，使用浏览器调试功能的 Network 查看请求的信息如图 11-2-6 所示。

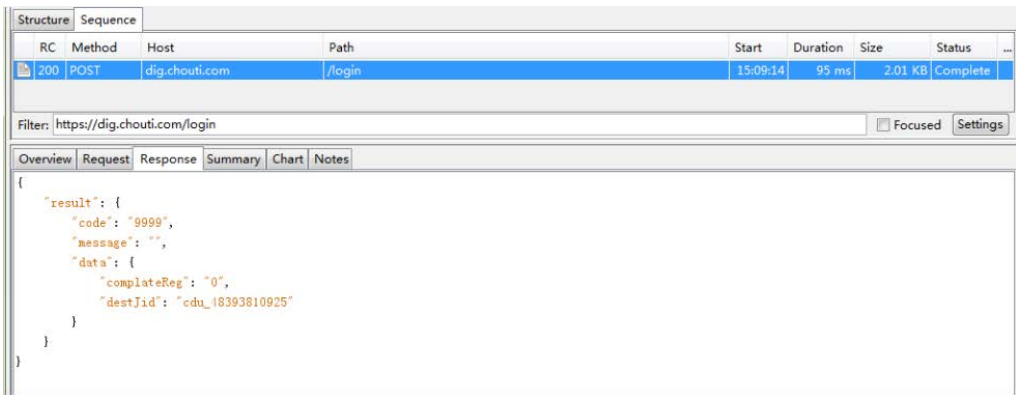


图 11-2-6

点击最后一个 login，具体请求信息为：

General 信息为:

```
Request URL: https://dig.chouti.com/login
Request Method: POST
Status Code: 200 OK
Remote Address: https://dig.chouti.com/
```

Response Headers 信息为:

```
Server: Tengine
Content-Type: text/plain; charset=utf-8
Content-Length: 96
Connection: keep-alive
Date: Thu, 31 Jan 2019 07:08:49 GMT
Cache-Control: no-cache
Set-Cookie: puid=6cb0ballbb026685f87ed7044c2d1d3a; domain=chouti.com; path=/; expires=Sat, 02-Mar-2019 07:08:49 GMT
Vary: Accept-Encoding
Via: cache31.l2nu16[60,0], cache2.cn451[81,0]
Timing-Allow-Origin: *
EagleId: 752222ca15489185290571526e
```

Request Headers 信息为:

```
Host: dig.chouti.com
Connection: keep-alive
Content-Length: 46
Accept: */*
Origin: https://dig.chouti.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36
DNT: 1
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://dig.chouti.com/
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN, zh;q=0.9
Cookie: gpsd=698471c95aff035be8d4869235bc6225;
gpId=ac6249d0fb574589acc5def913eb8753; _9755xjdesxxd_=32;
YD00000980905869%3AWM_TID=IdnDYQ0lDnBARERRVY9kBbtC%2BfqlyID; JSESSIONID=
aaahKsepcOhsja7oJdeIw; gdxidpyhxdE=Es1aP0%2BZkJkd6fmM2AHRONJXSpBDTG6%5CZb
qL9Ir60mMMyl0XGfjqJr43WjgOn%2FuMXmBEbY9GY2eCsrRBtS86u6Dka5fClNx0unrsN%2F
JtS32iGcsVBsD%5CRTYDENjBAWkjYXZRvu%5CCY%2BvseV4mxI7oysQ0lucMeMzRxBcLzW%5
CgwGu924IP%3A1548919186294; YD00000980905869%3AWM_NI=h%2B1VL%2B7pVhJdTs31
PxrGCFaWPOTWG4qsCus57rkW4%2FP%2BblGnVbUyoqSMf4Qwr3jrMikQsongSrgty7o9FTaT
```

```
nudJ0C4yxS011wn7q%2B0kqUB9APcABompS%2BDu0%2BCmVpdzWko%3D;YD0000098090586
9%3AWM_NIKE=9ca17ae2e6ffcd170e2e6eeaac13a83a6bba4d761b0b48ea2d54f869a8b
abbb68a1b4979bf25297f09c92f52af0fea7c3b92aacf5ffb7cf21f5b09a89c97098adfd
d2d374b2998da7c47d94919cb8c640b3ad8d96f23ff5aea598b72594f5ae8dc66696be9b
9aee7b9193aad5d47db2eda693d063a98fbd90ed7b91aea995fc3dade78ab5e460aeabac
abb349f8a7b8b4b368b0a9a2d8f421f4b5bfb6c1338390ada2c165afb09dccc77ff4f58d
8ed57ca8a6998cee37e2a3; puid=cdu_48393810925
```

Request Payload 信息:

```
phone:86134****5195
password:asd888
oneMonth:1
```

在以上代码中,可以得到如下的信息:

请求地址: <https://dig.chouti.com/login>

请求方法: POST

请求头信息:

Content-Type:application/x-www-form-urlencoded; charset=UTF-8

请求参数:

phone=86134****5195&password=asd888&oneMonth=1

接下来在 PostMan 中实现该过程, PostMan 的内容如图 11-2-7 所示。

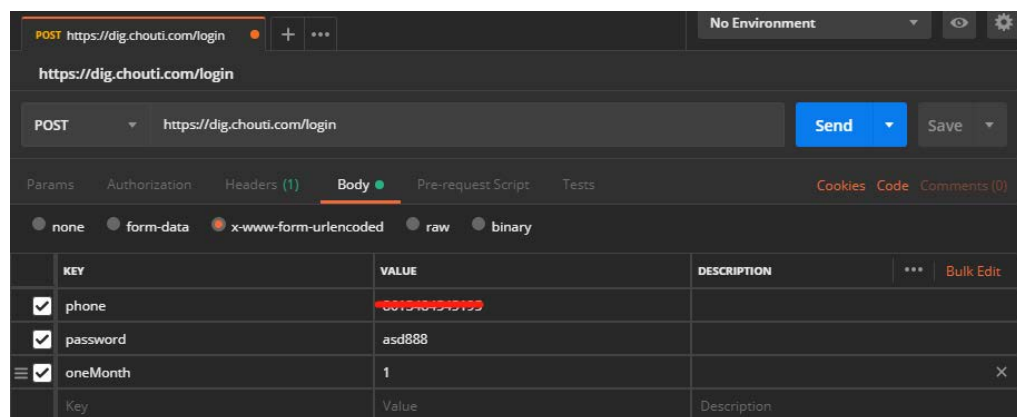


图 11-2-7

点击“Send”(发送)后,在 Body 中看到服务端返回的响应数据如图 11-2-8 所示。

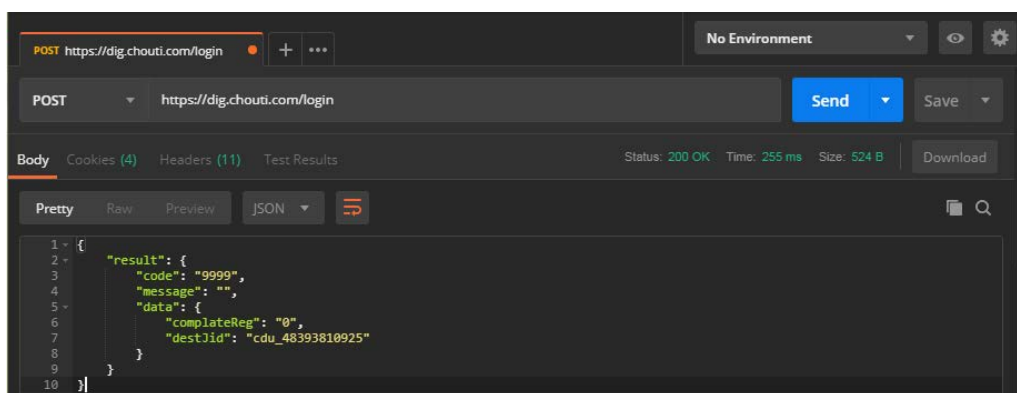


图 11-2-8

4. 接口的断言

在自动化测试中，每一个接口测试用例都需要断言，无断言的自动化测试用例是无效的。那么，针对以上请求，如何在 PostMan 中编写的断言呢？在 PostMan 中编写断言是在 Tests 中，在 Tests 中定义一个变量，把服务端返回的响应数据存储到一个变量中，然后进行断言的处理，Tests 编写的内容如图 11-2-9 所示。

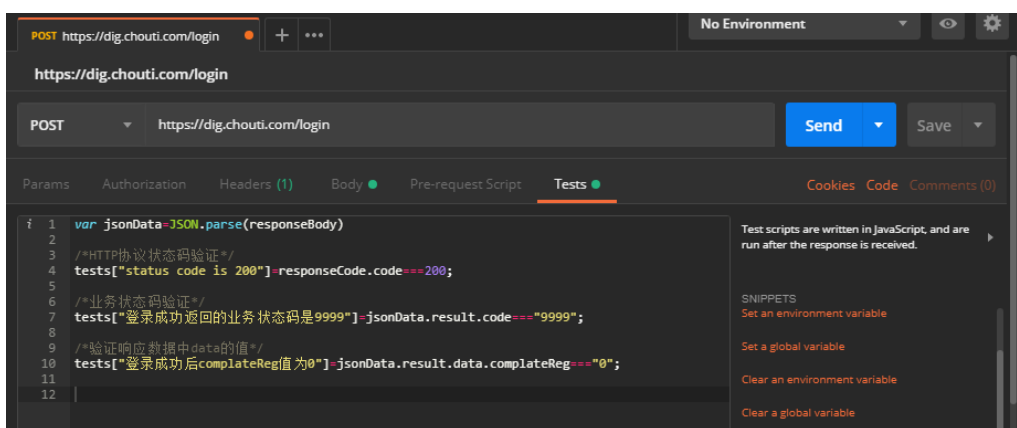


图 11-2-9

注解：在以上 Tests 中，断言主要有三部分，分别是 HTTP 的协议状态码，业务状态码，和登录成功后响应数据 data 中具体字段的值。把服务端返回的响应数据存储存储在变量 jsonData 中，然后断言业务状态码 code 是不是“9999”，该代码是 tests[“注释信息，也就是结果中打印的信息”]=获取 status 值=期望的值。

5. Token 的获取

在接口测试中，经常需要获取 Token，每一次登录成功后，服务端都会生成一个随机的字符串也就是 Token，然后把 Token 返回给客户端，客户端带着这个 Token，就可以操作服务端系统的业务。获取 Token 的步骤是：

- (1) 先请求登录接口；
- (2) 登录成功后，返回 Token；
- (3) 获取 Token；
- (4) 把获取的 Token 当作下一个接口的请求参数。

依据以上步骤，结合实例来看具体的实现过程。用 Charles 抓取 login 和 infoGet 接口的网络通信报文，如图 11-2-10 所示。

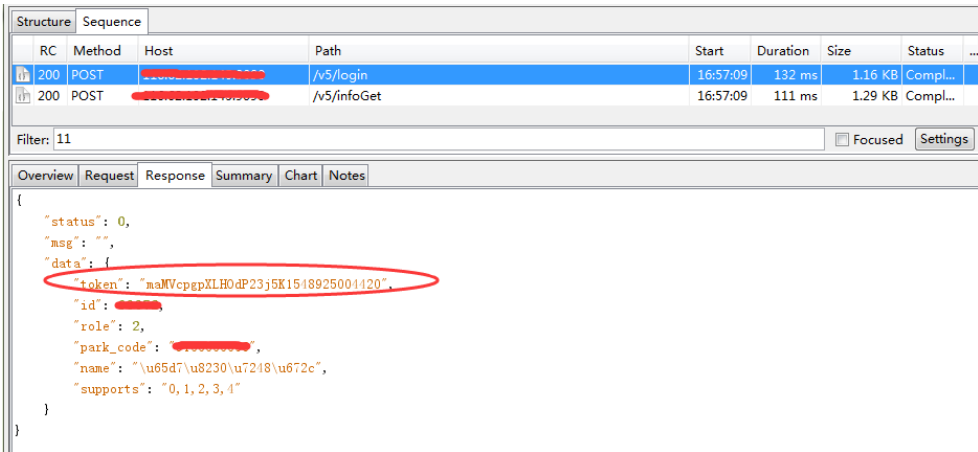


图 11-2-10

infoGet 接口的请求参数如图 11-2-11 所示。

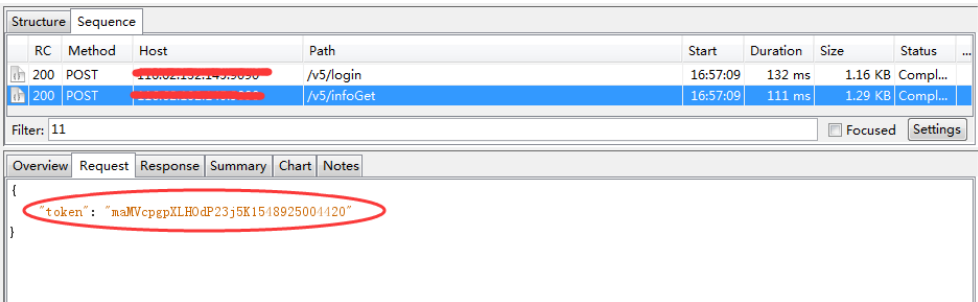


图 11-2-11

注解：在图 11-2-10 和图 11-2-11 中可以看到，login 接口登录成功后返回了 Token，接口 infoGet 请求参数中的 Token 必须与登录成功后返回的 Token 一致，如果不一致，请求无效。

下面通过 PostMan 实现该过程。首先创建 login 接口，在 Tests 中把返回的 token 定义成一个变量，在 infoGet 接口中调用这个变量，login 接口内容如图 11-2-12 所示。

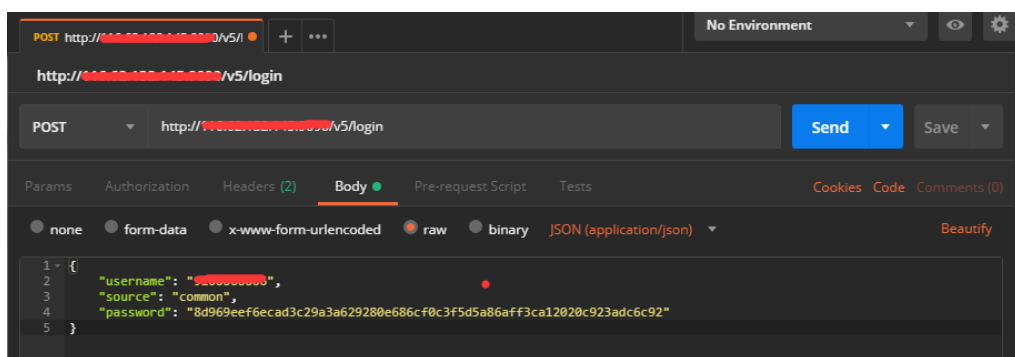


图 11-2-12

点击“Send”（发送）按钮后，返回的响应数据内容如图 11-2-13 所示。

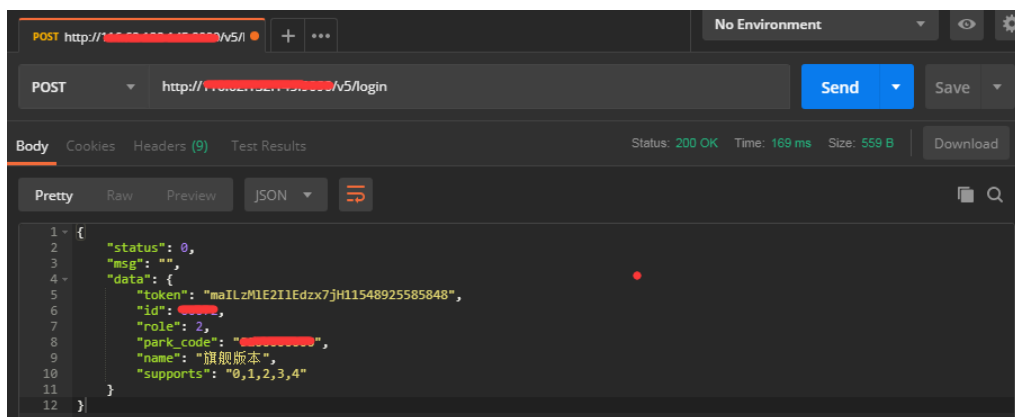


图 11-2-13

在 Tests 中把返回的 Token 定义一个变量存储到里面，如图 11-2-14 所示。



图 11-2-14

在登录成功后的响应数据中获取 token，并存储到变量 token 中，在 infoGet 的接口请求参数中调用变量 token，如图 11-2-15 所示。

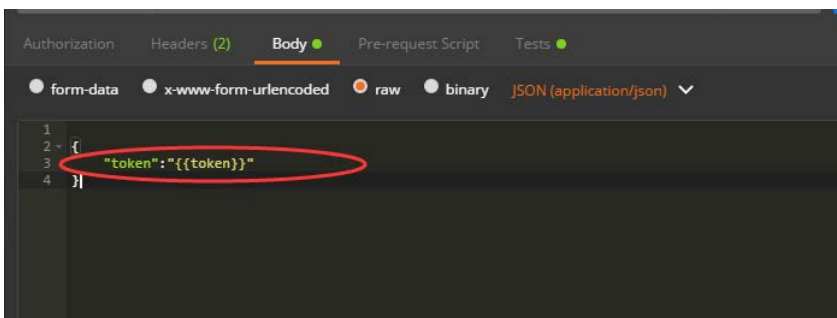


图 11-2-15

单独执行 infoGet 接口用例，执行的结果如图 11-2-16 所示。

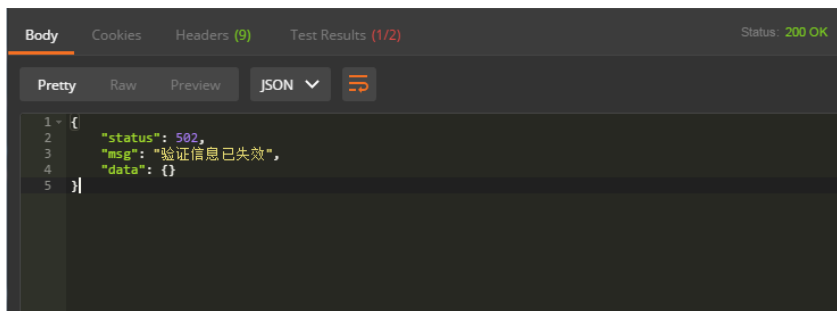



图 11-2-16

注解：单独执行 infoGet 接口之所以会出错，是因为请求参数中 token 只是调用了变量，由于没有执行 login 接口，所以也就无法获取登录成功后响应数据中的 token，也就是说 infoGet 接口请求参数中的 token 值与 login 登录成功后的 token 值不一致。

6. Collections 实战

在以上实例中单独执行 infoGet 的接口失败，导致 login 和 infoGet 的接口关联不起来。在 PostMan 中可以应用 Collections 来解决该问题，Collections 把所有的请求组合到集合中，使接口测试用例有顺序地执行。使用 Collections 一方面可以解决接口测试用例执行的顺序问题，另外一方面可以避免之前重复执行接口测试用例的烦琐，省去在历史记录中搜索已执行过的操作。在 PostMan 左边的 Collections 中，点击 ，在弹出的对话框中，填写 Name，然后点击“Create”，如图 11-2-17 所示。

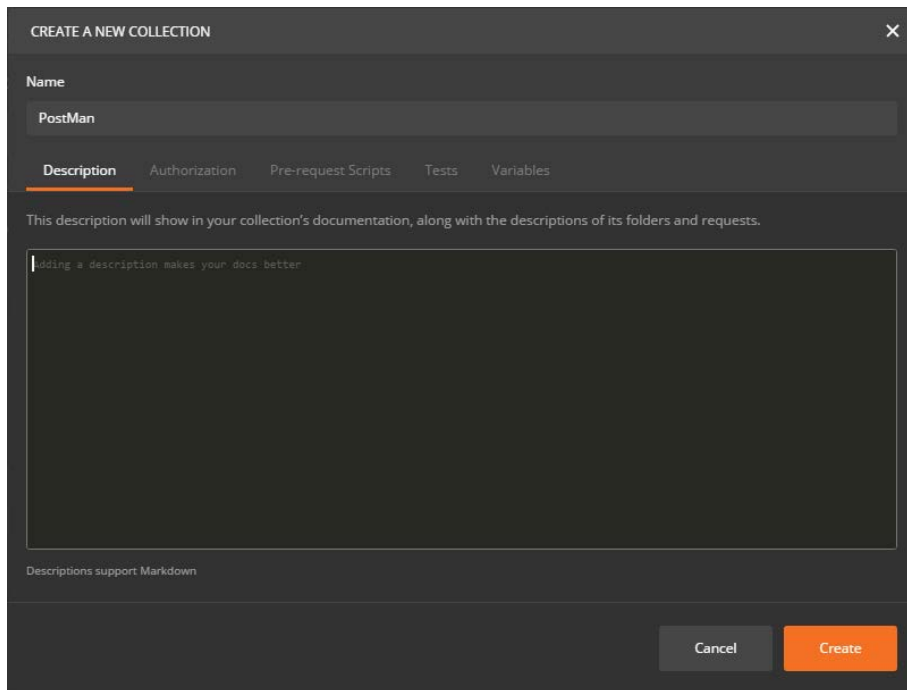


图 11-2-17

在接口用例 login 和 infoGet 中点击 Save，添加到集合 PostMan 中，添加后如图 11-2-18 所示。

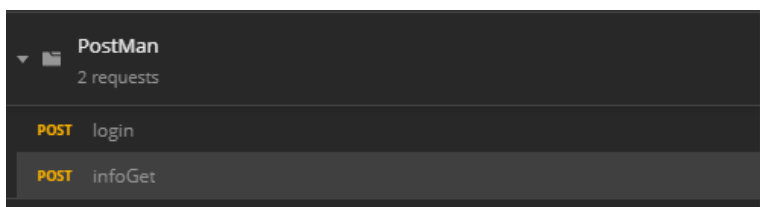


图 11-2-18

把接口测试用例添加到 Collections 中，然后执行 PostMan 中所有的接口用例，点击 PostMan 向右的箭头，在新的页面中点击“Run”按钮，如图 11-2-19 所示。

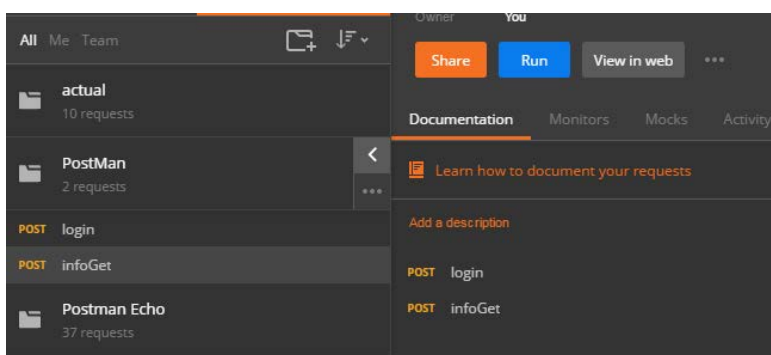


图 11-2-19

点击“Run”按钮后，打开一个新的页面，点击“Run PostMan”，可以看到接口用例执行的结果，如图 11-2-20 所示。

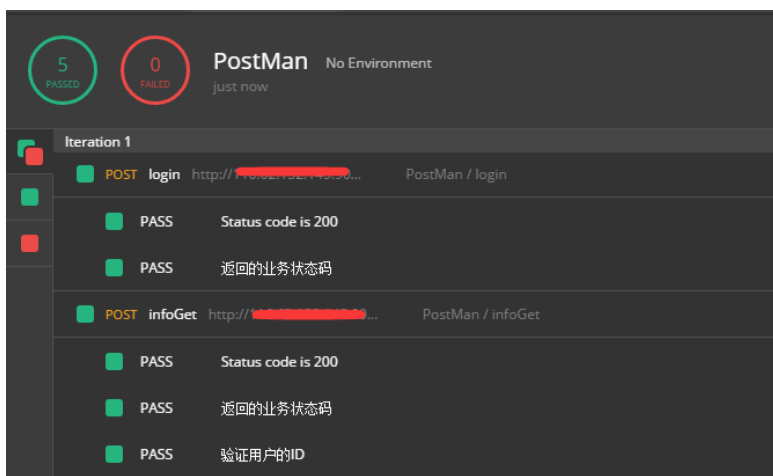


图 11-2-20

注解：在 PostMan 的执行结果中，通过数是依据用例的断言而来的，例如，一个用例有 N 个断言，N 个断言都验证通过，那么执行结果中 PASSED 就是 N。在这里 login 断言有两个，infoGet 断言有三个，执行后，login 和 infoGet 的接口断言都验证通过，所以 PASSED 显示 5，FAILED 显示 0。

7. Variables 的实战

在上一小节中我们学习了 Collection 的知识，也就是说把 login 和 infoGet 接口加入到了 PostMan 中，解决了执行顺序的问题，并且可以进行批量执行。但是在实际应用中请求地址往往是不确定的，IP 地址和请求端口会有改变，一旦改变，PostMan 中接口用例中的请求地址全部要修改，维护成本较高。这时可以使用 Collection 的 Variables。把公共数据分离到 Variables 中，在 Variables 中把公共数据定义成一个变量，将每个接口请求地址直接替换成变量就可以了，这样即使地址修改了，也只需要维护 Variables 中的数据。下面具体来看看实现的过程。

在 PostMan 集合中点击 **...** 出现下拉框，在下拉框中选择“Edit”选项，操作如图 11-2-21 所示。

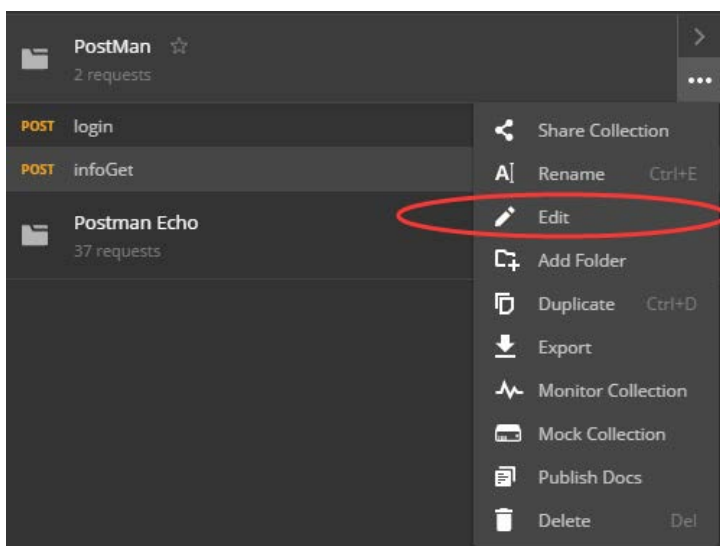


图 11-2-21

选择 Edit 选项后，在弹出页面中选择“Variables”选项，在 Variables 中，把请求地址、登录用户名和登录密码分离出来，如图 11-2-22 所示。

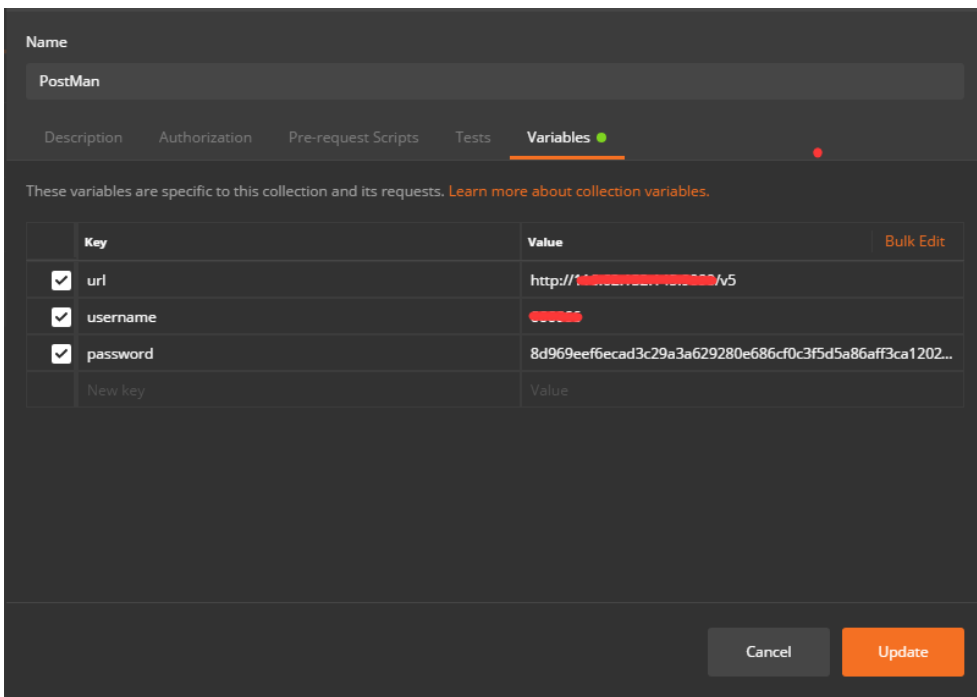


图 11-2-22

点击 Update 按钮完成修改，再次修改 login 接口，把请求地址，用户名和密码调用定义的变量，修改后如图 11-2-23 所示。

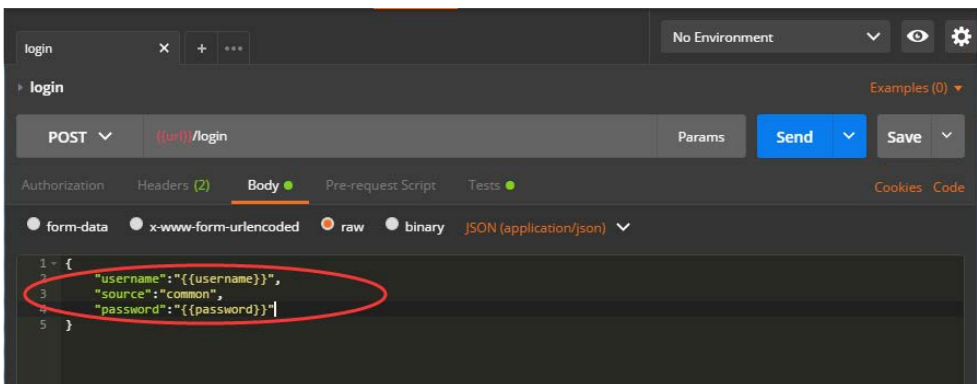


图 11-2-23

修改后的 infoGet 接口如图 11-2-24 所示。

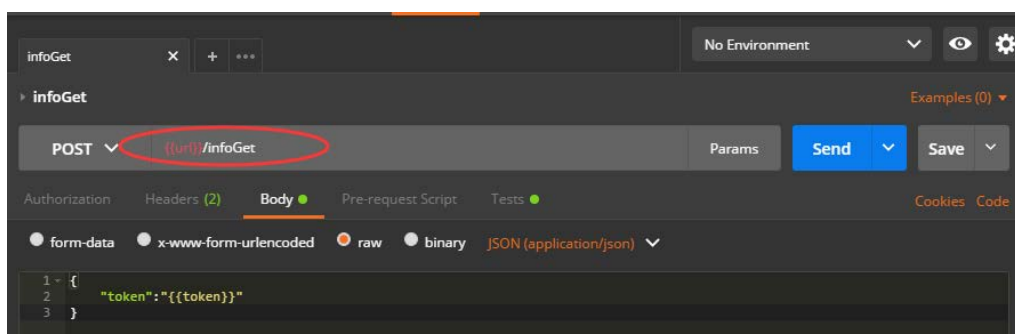


图 11-2-24

再次执行 PostMan，查看本次修改是否影响执行的结果。执行后结果正常，如图 11-2-25 所示。

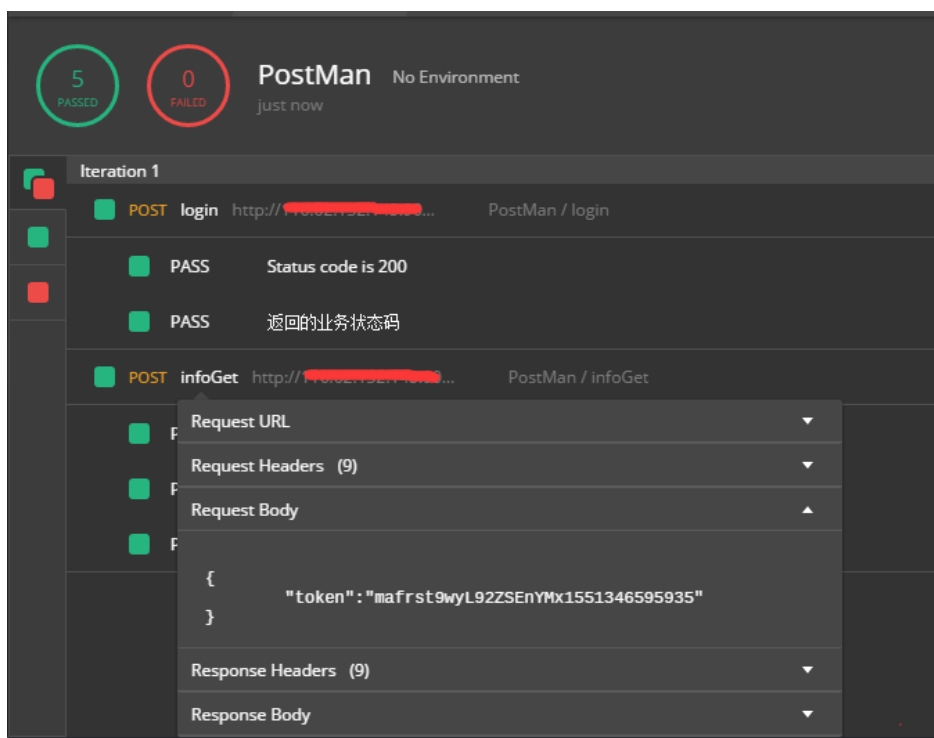


图 11-2-25

8. 请求带上 Cookie

并不是所有的程序都需要使用 Token 这种方式来处理，某些应用可使用

Session 来进行处理。例如，查看博客园我的粉丝，登录博客园成功后，点击“我的粉丝”，用 Charles 抓取“我的粉丝”接口的网络通信报文请求信息，如图 11-2-26 所示。

RC	Method	Host	Path	Start	Duration	Size	Status	...
200	GET	home.cnblogs.com	/u/weke/relation/followers	15:28:47	728 ms	5.42 KB	Complete	

Filter: home ☐ Focused

Overview	Request	Response	Summary	Chart	Notes
<pre> GET /u/weke/relation/followers HTTP/1.1 Host: home.cnblogs.com Connection: keep-alive Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 DNT: 1 Referer: https://home.cnblogs.com/u/weke/relation/schoolclasses Accept-Encoding: gzip, deflate, sdch, br Accept-Language: zh-CN,zh;q=0.8 Cookie: UM_distinctid=1626593019810e-023ea2cc2c7107-6b1b1279-100200-1626593019929; .CNBlogsCookie=41AE26C47146A4A1C3BD9... </pre>					

图 11-2-26

博客中我的粉丝显示如图 11-2-27 所示。



图 11-2-27

下面使用 PostMan 来请求“我的粉丝”接口，PostMan 的内容如图 11-2-28 所示。

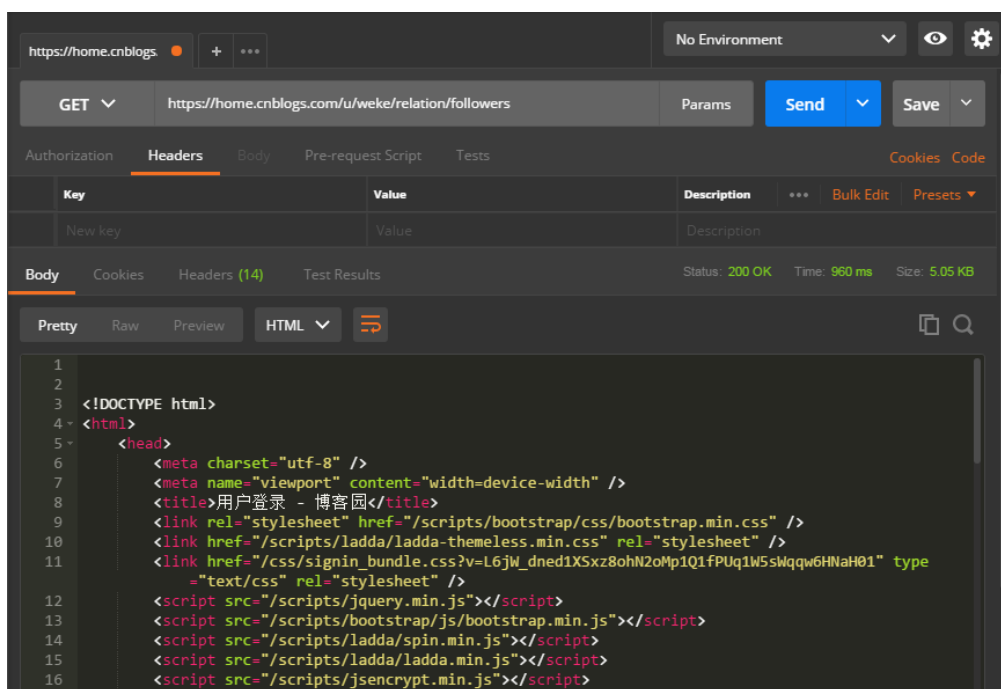


图 11-2-28

注解：请求后，系统直接跳转到博客园登录的页面，而没有返回“我的粉丝”，这是因为“我的粉丝”接口请求的时候没有带上 Cookie，所以请求认定用户是在未登录的情况下操作的，所以跳转到了登录的地方。

在 Charles 中复制 https://home.cnblogs.com/u/weke/relation/followers 请求时的 Cookie，添加到请求头中，添加后如图 11-2-29 所示。

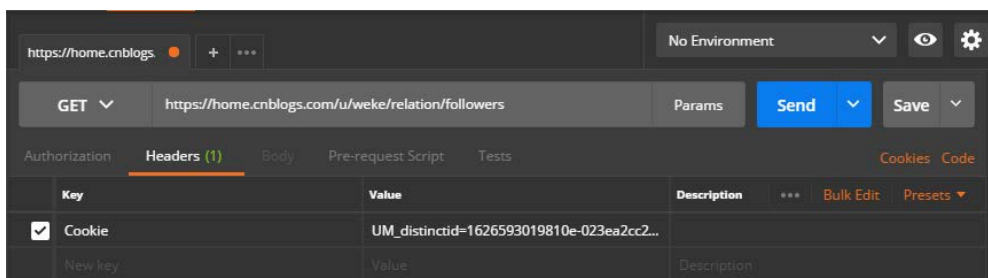


图 11-2-29

再次点击“Send”（发送）按钮，服务端返回的内容如图 11-2-30 所示。

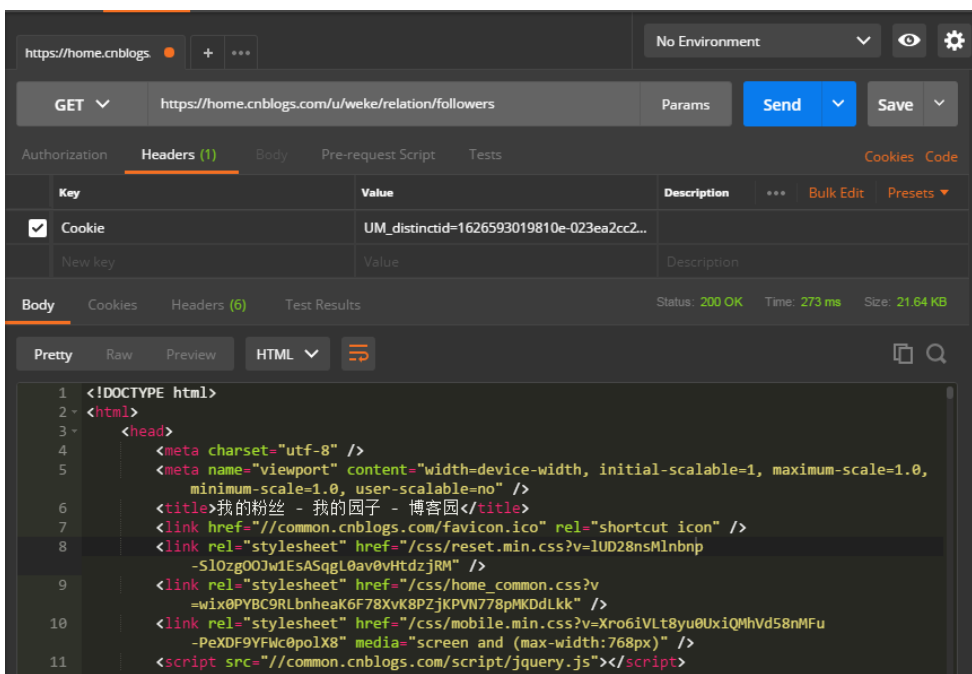


图 11-2-30

9. 鉴权的实战

鉴权（Authentication）是指验证用户是否拥有访问系统的授权。现在通过实例，来看一下使用 PostMan 测试工具是如何处理鉴权的，实例代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

from flask import Flask, jsonify, request, abort, url_for, make_response
from flask_httpauth import HTTPBasicAuth

u'''基于 flask 的 web service 的实例代码'''
app=Flask(__name__)

u'''增加 HTTPBasicAuth 的权限验证机制'''
auth=HTTPBasicAuth()

@auth.get_password
def get_password(username):
    if username=='wuya':
```

```

        return 'admin'
    return None

@auth.error_handler
def unauthorized():
    return make_response(jsonify({'error': 'Unauthorized access'}), 401)

datas=[
    {
        'userid':1,
        'username':u'李四',
        'identity card': '23012919950425723X',
        'room number': '1104',
        'phone': '13484545190',
        'vpl':u'京AJ3585',
        'check in': '2018-03-08 08:20:10',
        'check out': '2018-03-09 14:00:00'
    }
]

@app.route('/hotel/username/', methods=[ 'GET' ])
@auth.login_required
def show():
    return jsonify({'datas':datas})

if __name__ == '__main__':
    app.run(debug=True)

```

执行以上程序后，在 PostMan 中输入 `http://localhost:5000/hotel/username`，点击“Send”（发送）按钮后，返回的结果如图 11-2-31 所示。

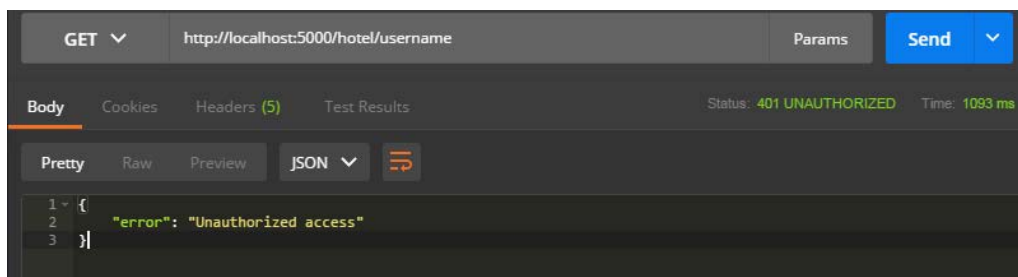


图 11-2-31

从图 11-2-30 中可以看到 HTTP 返回的状态码是 401 的错误，401 指的是对被请求的页面需要用户名和密码，查看服务端返回的头信息，如图 11-2-32 所示。

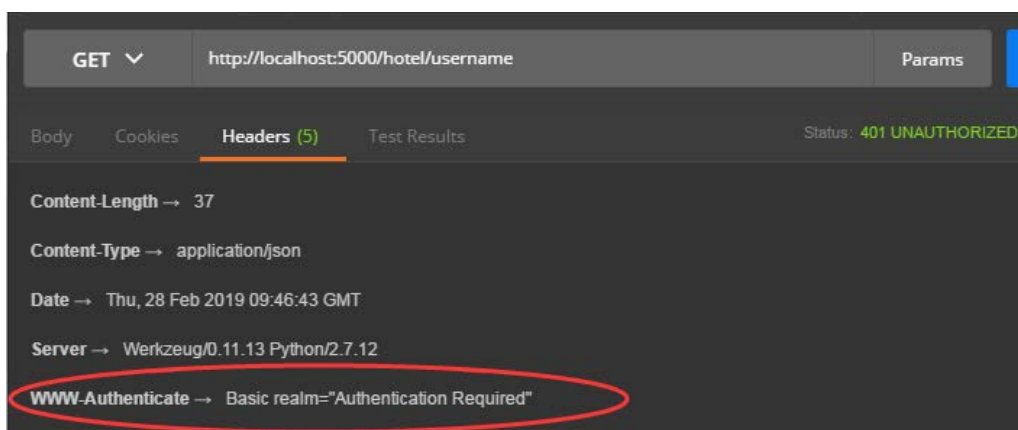


图 11-2-32

依据以上 Headers 信息，可以得到请求失败的原因，因为没有做鉴权的处理，在本实例中鉴权采用的是 HTTP 的基本认证方式，基本认证是一种允许 WEB 浏览器或其他客户端程序在请求时提供身份验证的一种登录方式，基本认证方式是一种流行的，行业标准的身身份验证方案，其包含的流程是：如果请求的资源需要身份验证，服务器将返回 HTTP 401 包含 WWW 身份验证消息头的响应；如果用户已使用 ID 和密码发送请求，在请求头中包含了授权的标识，服务器处理提交的凭据并提供访问权限。

依据以上代码可以得到用户名和密码分别是 wuya 和 admin，在 PostMan 中对该请求进行鉴权的处理，在 Authorization 的 TYPE 中选择“Basic Auth”，在 Username 和 Password 输入框分别输入 wuya、admin，输入的内容如图 11-2-33 所示。

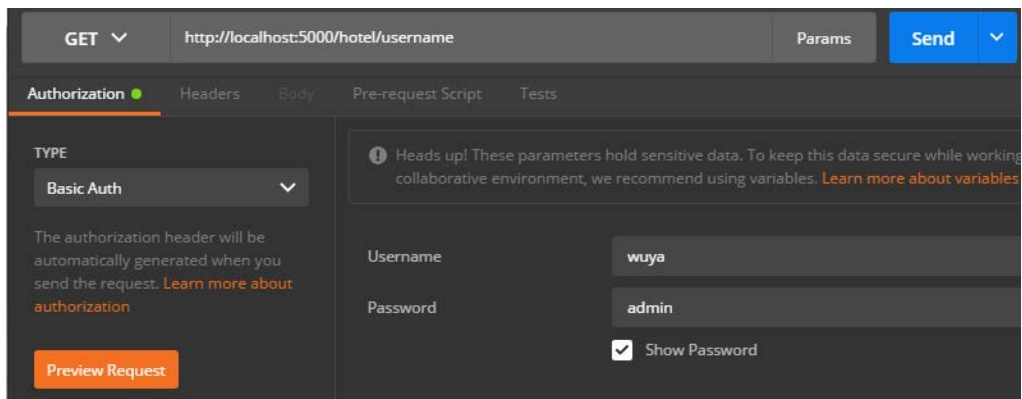


图 11-2-33

点击“Preview Request”后，在 Headers 中会新增 Authorization，如图 11-2-34 所示。

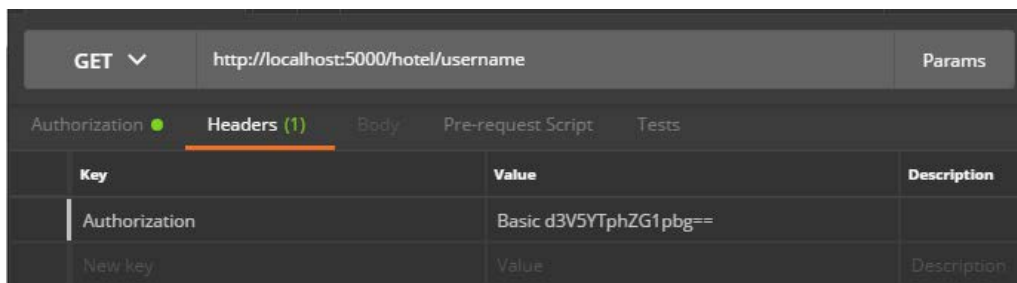


图 11-2-34

再次点击“Send”（发送）按钮，服务端返回的 HTTP 状态码和响应数据如图 11-2-35 所示。

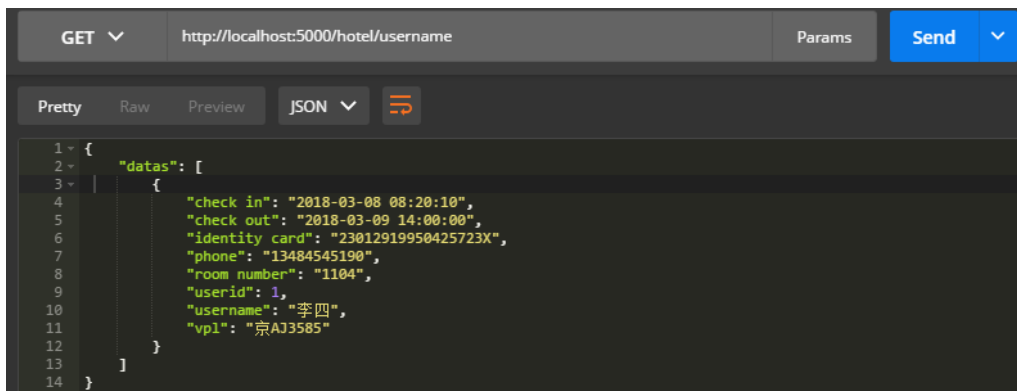


图 11-2-35

10. NewMan 的应用

使用 NewMan 可以轻松地运行和测试 PostMan 中 Collections 的接口用例，内置的可扩展性，可以轻松地集成到持续集成服务器和构建系统。安装 NewMan 前必须先安装 node.js，在安装 node.js 后，打开 cmd 命令提示符，使用如下命令安装 NewMan：

```
npm install -g newman --registry=https://registry.npm.taobao.org
```

NewMan 的安装如图 11-2-36 所示。

```

Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>npm install -g newman --registry=https://registry.npm.taobao.org
npm WARN deprecated node-uuid@1.4.8: Use uuid module instead
C:\Users\Administrator\AppData\Roaming\npm\newman -> C:\Users\Administrator\AppData\Roaming\npm\node_modules\newman\bin\newman.js
+ newman@3.9.3
updated 3 packages in 26.58s

C:\Users\Administrator>_

```

图 11-2-36

在这里，结合 PostMan 中的集合 PostMan，来介绍 NewMan 在 PostMan 工具中的应用。

点击集合 PostMan 的向左箭头，在弹出的界面中点击 Share，选择 Collection Link，点击“Get Link”，如图 11-2-37 所示。

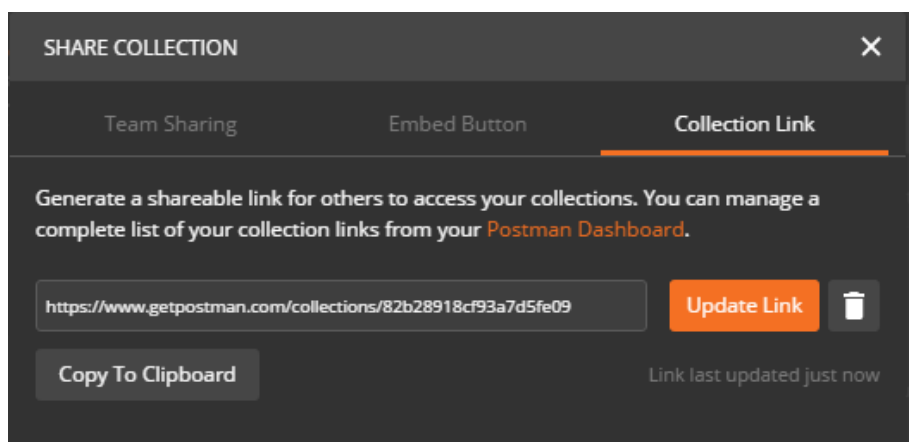



图 11-2-37

点击 Copy To Clipboard，在 cmd 的命令提示符中执行命令：

newman run https://www.getpostman.com/collections/82b28918cf93a7d5fe09

即可看到执行的结果。

还可以导出 Collections，然后使用 NewMan 执行导出的文件，点击集合 PostMan 后，在下拉框中点击“Export”，把导出的文件放在 C 盘的根目录下，如图 11-2-38 所示。

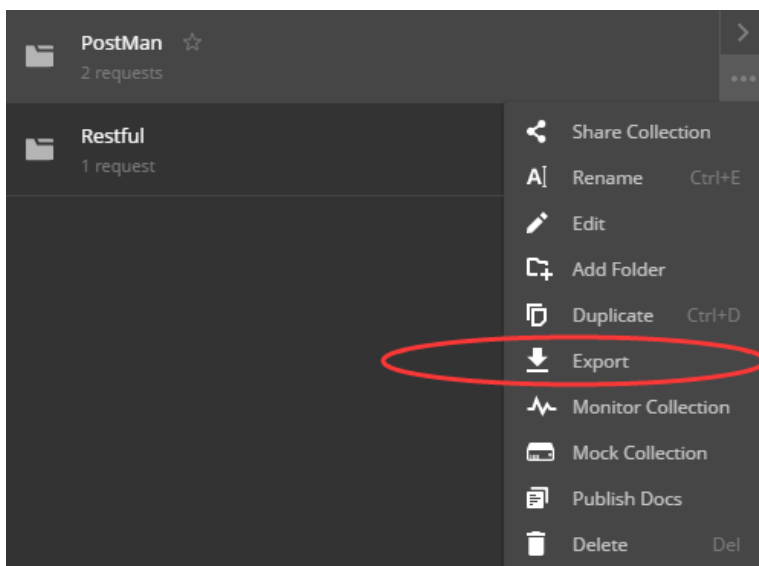


图 11-2-38

点击 Export，在弹出框中点击 Export，如图 11-2-39 所示。

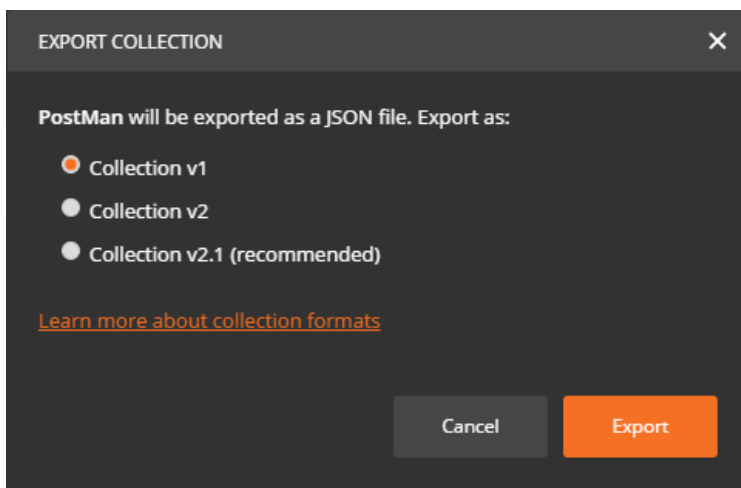


图 11-2-39

点击 Export 后，把文件存储在 C 盘的根目录下，如图 11-2-40 所示。

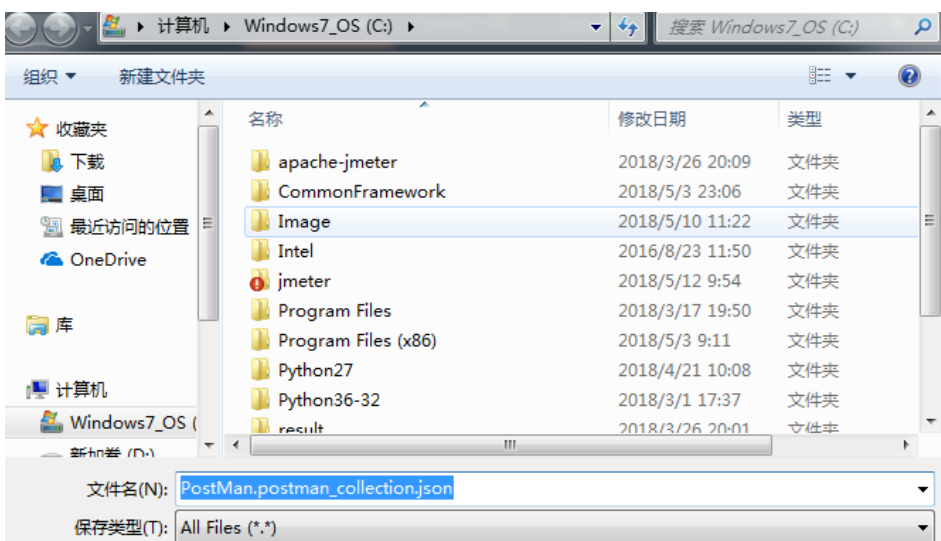


图 11-2-40

点击“保存”按钮后，在 cmd 的命令提示符下执行该文件，执行的命令为：

newman run PostMan.postman_collection.json

执行后屏幕显示如图 11-2-41 所示。

```
c:\>newman run PostMan.postman_collection.json
newman: Newman v4 deprecates support for the v1 collection format
Use the Postman Native app to export collections in the v2 format

newman
PostMan

→ login
POST http://[redacted]/v5/login [200 OK, 545B, 219ms]
✓ Status code is 200
✓ 返回的业务状态码

→ infoGet
POST http://110.02.152.113:7878/v5/infoGet [200 OK, 989B, 93ms]
✓ Status code is 200
✓ 返回的业务状态码
```

图 11-2-41

注解：如图 11-2-42 为执行命令后系统显示的详细信息。

	executed	failed
iterations	1	0
requests	2	0
test-scripts	4	0
prerequisite-scripts	2	0
assertions	5	0
total run duration: 515ms		
total data received: 413B <approx>		
average response time: 172ms		

图 11-2-42

但这份测试报告看起来并不友好，接下来，我们可调用 NewMan 中的参数 reports 生成一份基于 HTML 的测试报告，执行命令如图 11-2-43 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd c:/

c:\>newman run Postman.postman_collection.json --reporters html

c:\>
```

图 11-2-43

执行后，会在 C 盘下自动创建一个 NewMan 的文件夹，并在该文件下生成一份基于 HTML 的测试报告，如图 11-2-44 所示。

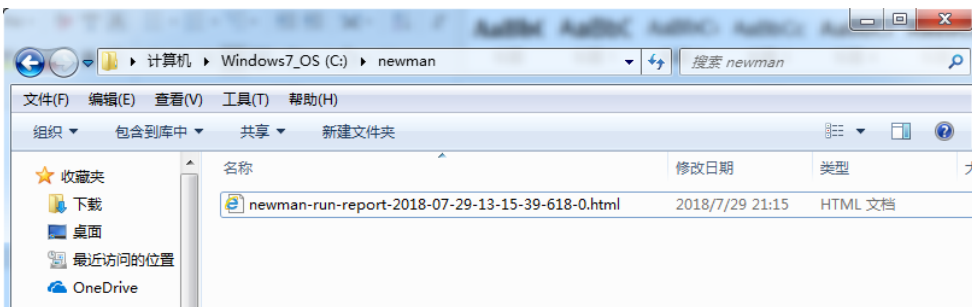


图 11-2-44

打开该 HTML 文件便会看到详细的执行结果，如图 11-2-45 所示。



图 11-2-45

具体接口测试用例执行的结果如图 11-2-46 所示。

Requests

http://180.97.80.42:9090/v4/login			
Method	POST		
URL	http://180.97.80.42:9090/v4/login		
Mean time per request	232ms		
Mean size per request	110B		
Total passed tests	3		
Total failed tests	0		
Status code	200		
Tests	Name	Pass count	Fail count
	Status code is 200	1	0
	返回的业务状态码是0	1	0
	获取token成功	1	0

图 11-2-46

11. NewMan+Jenkins 的应用

下面结合 NewMan 使接口测试用例在 Jenkins 平台上执行。首先在 Jenkins 平台上创建一个自由风格的 Job 名称为 PostMan，然后在增加构建步骤中选择“Windows batch command”并编写执行命令，如图 11-2-47 所示。



图 11-2-47

构建后操作选择“Publish HTML reports”，在 Reports 中填写测试报告的目录，如图 11-2-48 所示。

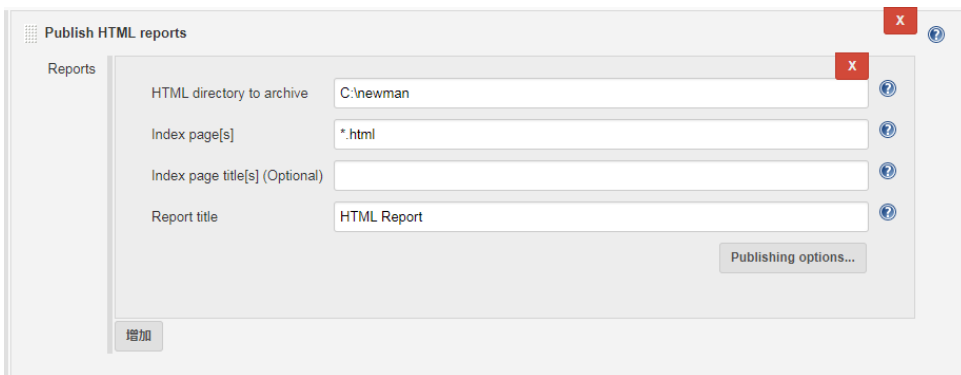


图 11-2-48

保存后，选择“立即构建”选项，就会生成对应的测试报告，输出的结果如图 11-2-49 所示。



图 11-2-49

12. PROXY SETTINGS

在 PostMan 测试工具中，可以通过设置 PROXY SETTINGS 来抓取手机上的请求，这样，在手机上的请求就可以在 PostMan 中看到。在 PostMan 中点击雷达状的图标，弹出 PROXY SETTINGS 配置对话框，如图 11-2-50 所示。

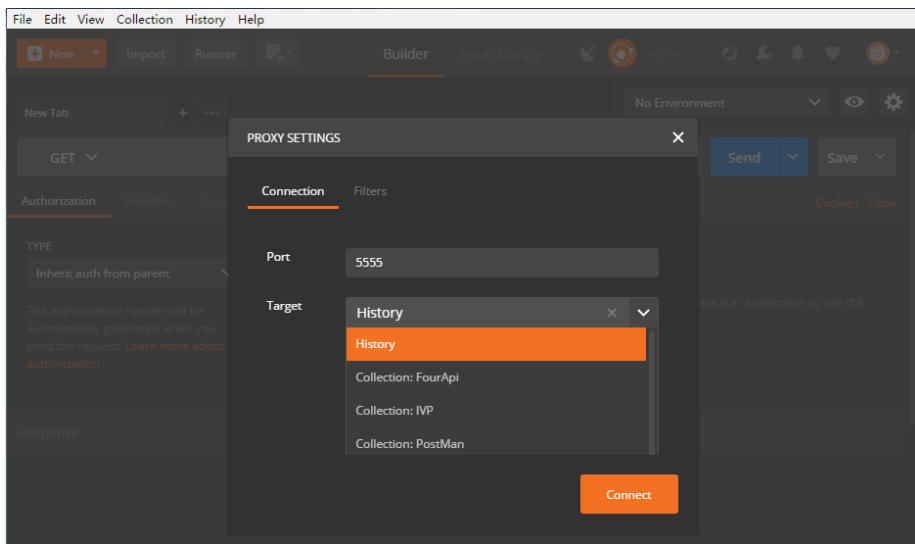


图 11-2-50

点击“Connect”，提示 Proxy Connected，如图 11-2-51 所示。

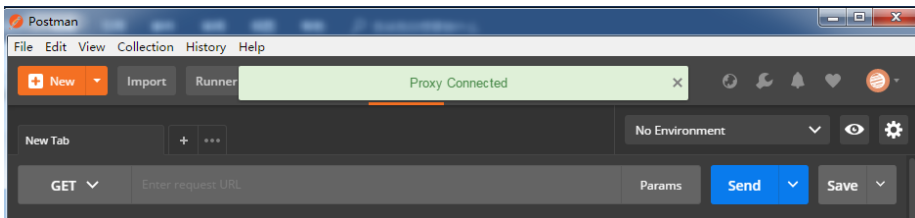


图 11-2-51

务必要保持抓包的手机和电脑是在同一个网络下，在手机中配置代理，地址填写电脑的 IP 地址，端口填写 5555。手机配置成功后，在手机上打开 app 的应用程序后，PostMan 的 history 就会抓取到手机的操作记录，如图 11-2-52 所示。

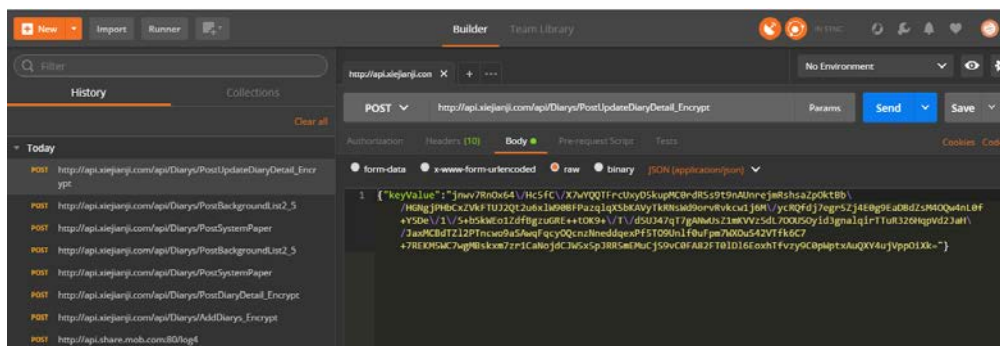


图 11-2-52

11.3 PostMan 接口测试实战

这里通过一个完整的业务来说明 PostMan 测试工具在接口自动化测试中的应用。业务要求是登录到某系统，创建用户，然后，对用户进行冻结和激活操作，最后删除用户。在 PostMan 中创建名为 actual 的 Collections，如图 11-3-1 所示。

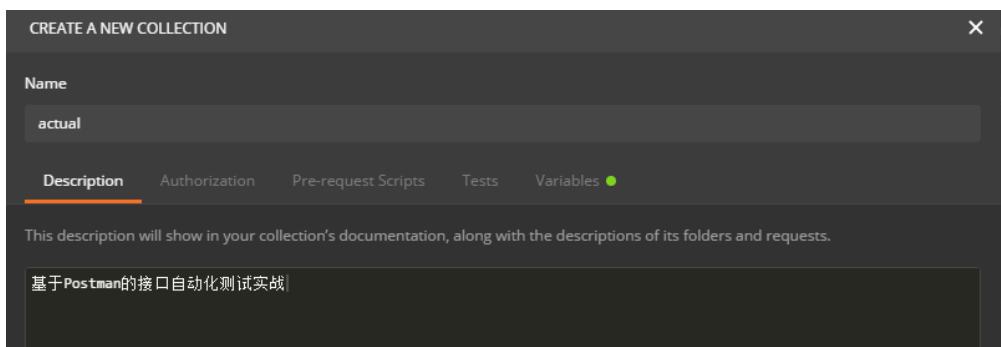


图 11-3-1

在 Variables 分离出公共数据，主要是请求地址，登录系统的账号和密码，如图 11-3-2 所示。

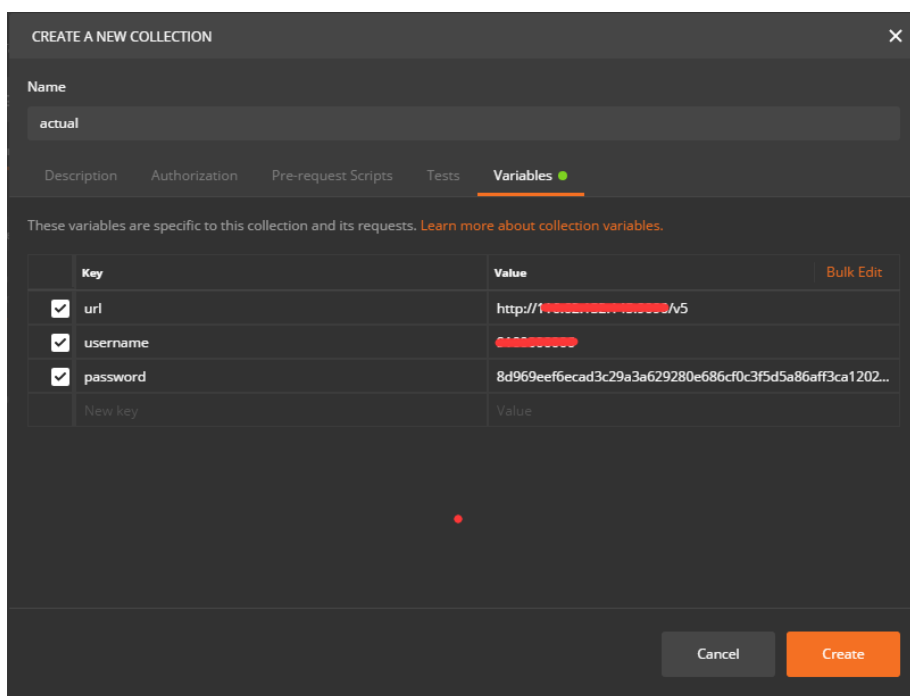



图 11-3-2

点击“Create”按钮创建集合。在集合 actual 中点击 ，在弹出的下拉框中点击“Add Folder”、分别添加登录、创建用户、用户列表、冻结用户、激活用户、删除用户的 Folder，如图 11-3-3 所示。

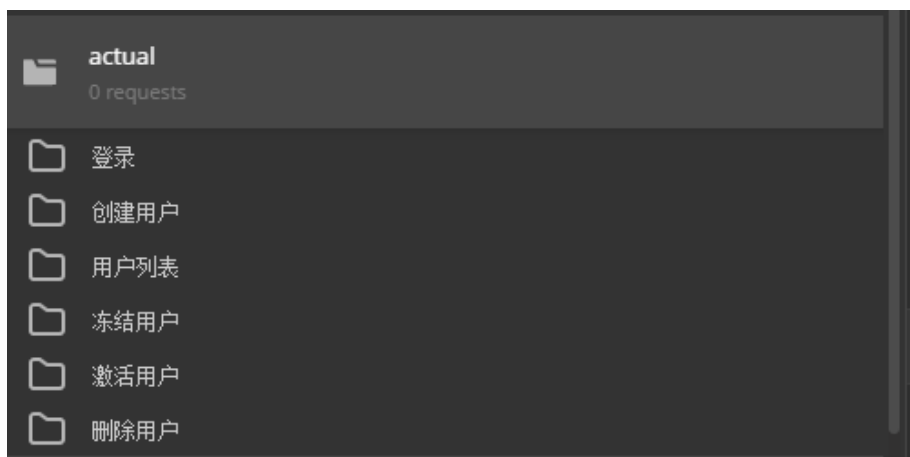


图 11-3-3

在登录的 Folder 中添加登录的接口用例，分别是 login 和 infoGet，login 接口用例的内容如图 11-3-4 所示。

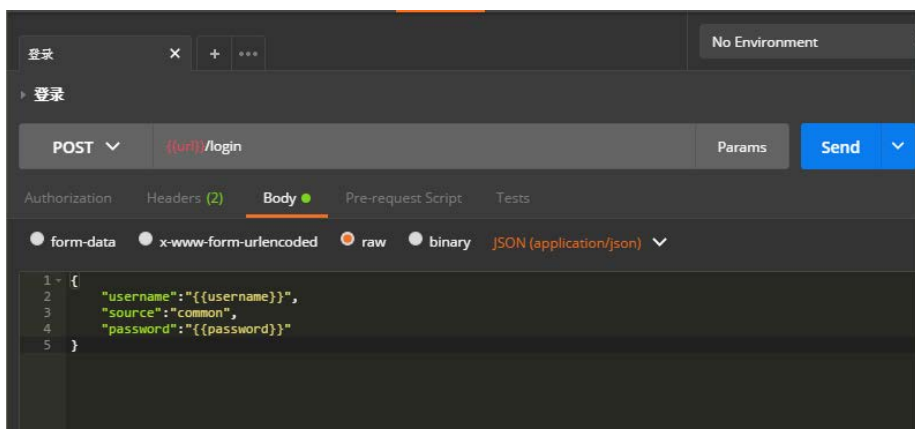


图 11-3-4

注解：前面已经把请求地址和登录系统的账号分离到了 Variables 中，所以这里只需要直接调用就可以了。

点击 Send 后，返回响应数据。在响应数据中包含有 token，需要把 token 存储在一个变量中，Tests 的代码如图 11-3-5 所示。

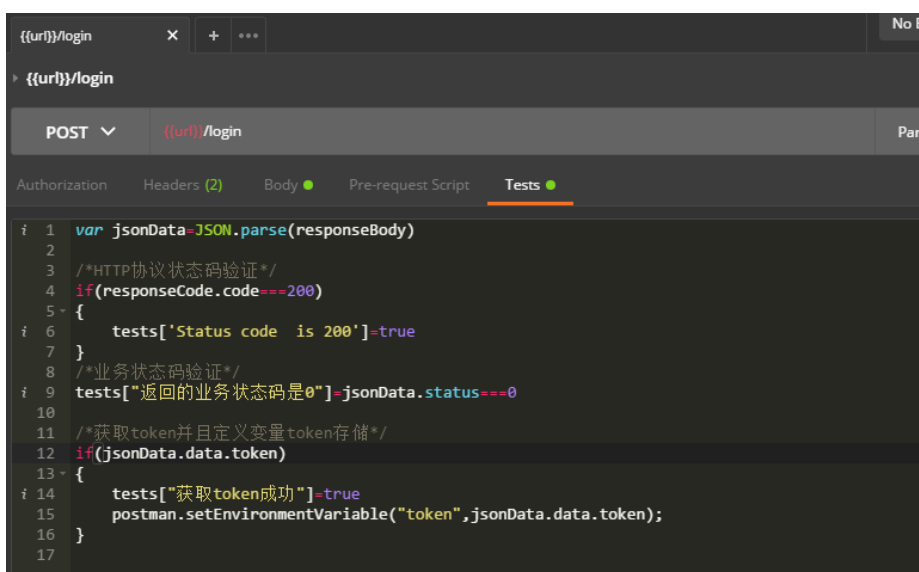


图 11-3-5

把 login 保存到 actual 下的登录中，再将 infoGet 的接口添加到登录中，登录的 Folder 中的接口用例如，图 11-3-6 所示。



图 11-3-6

接下来编写添加用户的接口用例。添加用户后，会返回用户的 ID。在 Tests 中获取用户的 ID，放在变量 userID 中，如图 11-3-7 所示。

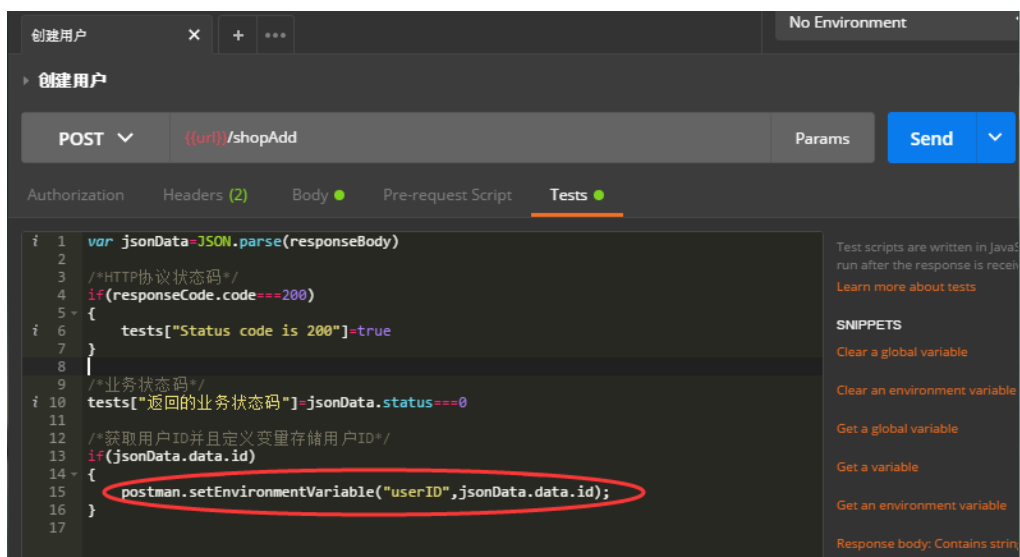


图 11-3-7

注解：用户登录成功后，在返回的响应数据中获取用户的 ID，因为后面进行的删除等操作会使用到用户的 ID。

继续完善用户列表的用户查询接口用例。对接口测试的名称进行修改，修改后的登录，添加用户，查询用户的接口用例例如，图 11-3-8 所示。

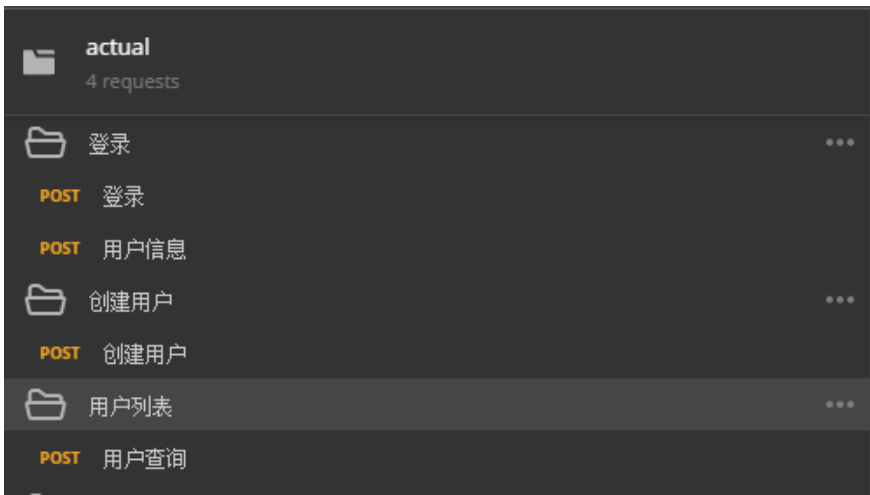


图 11-3-8

接下来完善冻结用户的用例。冻结用户后，需查询用户并验证用户的状态，因为冻结用户后，并没有返回用户的状态，验证冻结用户的断言验证如图 11-3-9 所示。

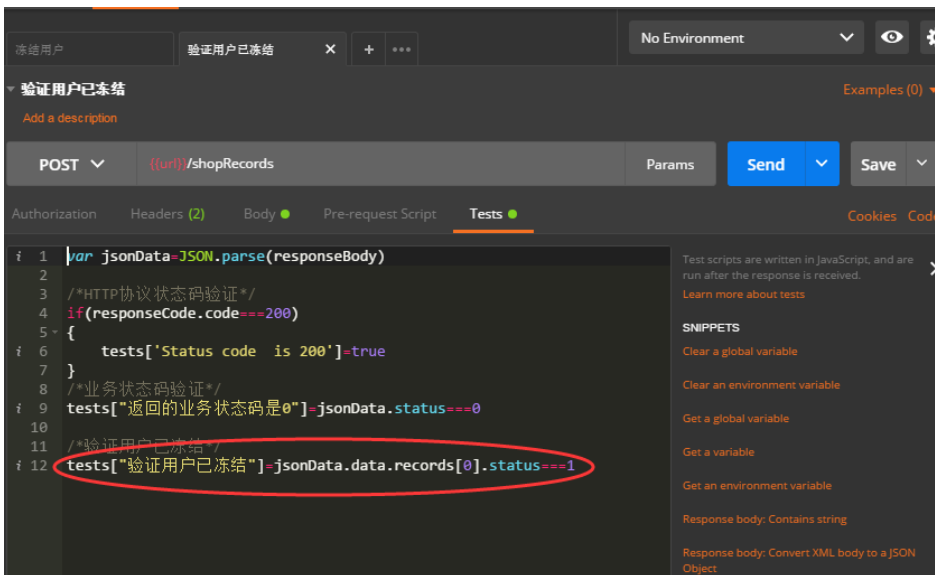


图 11-3-9

注解：冻结用户后，并没有返回用户的状态，无法确定用户状态是否为冻结的状态。因此需要通过查询用户，来验证用户的状态是否为冻结的状态，也就是

status 字段的值是否为“1”。一个接口测试用例执行成功不代表它的业务是正确的，因此每一个接口测试用例都要进行严格的断言验证。

接着完善激活用户的用例。激活用户与冻结用户一样，激活后，通过用户列表来验证用户的状态。验证激活用户的断言如图 11-3-10 所示。

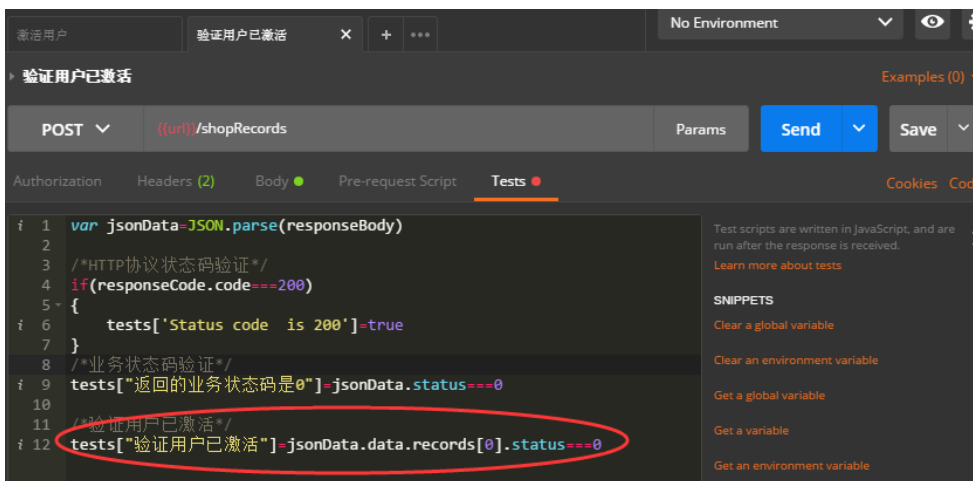


图 11-3-10

接着完善删除用户的接口用例。删除用户后，在用户列表中查询用户，如果查询的结果是 0，那么可以认为用户已删除，验证用户已删除的接口断言验证如图 11-3-11 所示。

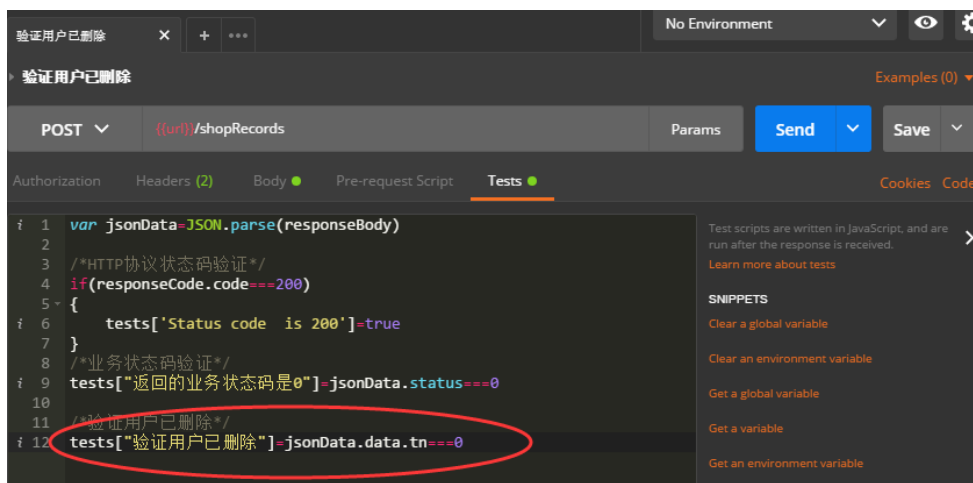
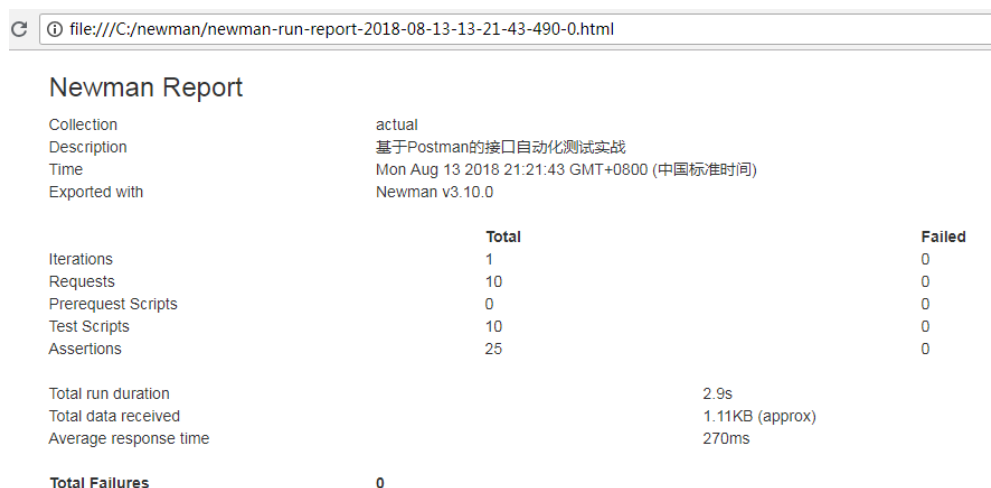


图 11-3-11

已完成所有的接口用例，将 `actual` 导出，导出的文件名为 `actual.postman_collection.json`，将文件保存在 C 盘下。接下来使用 `NewMan` 执行该文件，生成基于 `HTML` 的测试报告，执行的命令为：

```
newman run actual.postman_collection.json --reporters html
```

执行成功后，在 `newman` 文件夹下打开 `html` 显示测试结果，如图 11-3-12 所示。



Newman Report		
Collection	actual	
Description	基于Postman的接口自动化测试实战	
Time	Mon Aug 13 2018 21:21:43 GMT+0800 (中国标准时间)	
Exported with	Newman v3.10.0	
	Total	Failed
Iterations	1	0
Requests	10	0
Prerequisite Scripts	0	0
Test Scripts	10	0
Assertions	25	0
Total run duration		2.9s
Total data received		1.11KB (approx)
Average response time		270ms
Total Failures	0	

图 11-3-12

使用 `PostMan` 测试工具，一方面可以获取到服务端响应的数据，方便与开发人员交流，另外一方面如果代码能力不强，也可以使用 `PostMan` 测试工具进行产品的接口自动化测试，从而提高测试效率。

第 12 章 JMeter 接口测试应用

JMeter 是一款纯 Java 语言开发的开放代码的应用程序，主要用来测试程序的性能，它最初被设计用于测试 Web 应用程序，后来扩展到其他领域的测试。

12.1 JMeter 简述

JMeter 是由 Apache 软件基金会的 Stefano Mazzocchi JMeter 开发的，后来又重新设计 JMeter 增强的图形用户界面和添加功能测试能力。JMeter 可以应用到很多测试场景，例如，性能测试、分布式性能测试、功能测试、接口测试等。本章节主要介绍的是应用于接口测试。JMeter 最新版本是 5.0（本书实例都是基于 JMeter4.0 进行）。JMeter 下载地址为：http://jmeter.apache.org/download_jmeter.cgi。下载 JMeter 后，JMeter 运行需要在 JDK 的环境下，JDK 环境的搭建这里不再介绍。

12.2 JMeter 的语言切换

搭建好 JDK 环境后，解压下载的文件，执行 bin 目录下的 jmeter.bat，JMeter 启动成功就会显示 GUI 界面，但是，此时 GUI 界面是英文的，切换到中文界面的操作为：点击 Options 下的 Choose Language，选择 Chinese(Simplified)，GUI 界面就会以中文显示，但是关闭 JMeter 后再次启动时，界面依然显示英文。为了使其启动即显示中文界面，可在 bin 目录下，编辑 jmeter.properties 文件，把 language=en 修改为 language=zh_CN，保存该文件，下次启动 JMeter 后，默认显示的就是中文了。

12.3 JMeter 的插件安装

在实际应用中，会遇到某些功能需要为 JMeter 单独安装插件，安装插件的

方式为下载 jmeter-plugins-manager-1.3.jar 文件，把该文件放到 JMeter 的 lib/ext 目录下。再次启动 JMeter 后，点击菜单栏的“选项”，可以看到“Plugins Manager”选项，点击“Plugins Manager”会弹出一个新的界面，如图 12-3-1 所示。

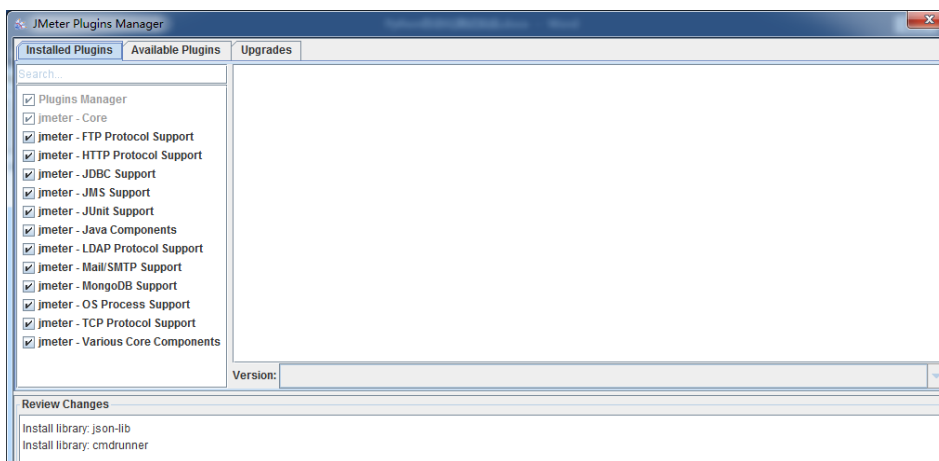


图 12-3-1

点击“Available Plugins”，选择要安装的插件，这里主要选择 JSON Plugins, PerfMon, XML Plugins, WebSocket 等插件，选中要安装的插件后，点击右下角的“Apply Changes and Restart JMeter”，就会自动下载插件并且安装，安装成功后系统自动重新启动 JMeter。

12.4 WebServices 的请求

在 JMeter 中创建计划和线程组，线程组名称是“WebServices 的请求”，用鼠标右键点击线程组名称，在配置元件中选择 HTTP 信息头管理器，在 HTTP 信息头管理器中添加消息头，消息头为 Content-Type:text/xml; charset=utf-8。这里请求的接口是查询电话号码，接口名称是 getMobileCodeInfo，请求参数和相应内容如下：

```
POST /WebServices/MobileCodeWS.asmx HTTP/1.1
Host: ws.webxml.com.cn
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://WebXml.com.cn/getMobileCodeInfo"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfo xmlns="http://WebXml.com.cn/">
      <mobileCode>string</mobileCode>
      <userID>string</userID>
    </getMobileCodeInfo>
  </soap:Body>
</soap:Envelope>
```

HTTP/1.1 200 OK

Content-Type: text/xml; charset=utf-8

Content-Length: length

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfoResponse xmlns="http://WebXml.com.cn/">
      <getMobileCodeInfoResult>string</getMobileCodeInfoResult>
    </getMobileCodeInfoResponse>
  </soap:Body>
</soap:Envelope>
```

用鼠标右键点击线程组，点击添加中的 **Sampler**，新建 HTTP 请求，方法选择 **POST**，路径填写：

<http://ws.webxml.com.cn/WebServices/MobileCodeWS.aspx?op=getMobileCodeInfo>。

在 **Body Data** 中填写如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMobileCodeInfo xmlns="http://WebXml.com.cn/">
      <mobileCode>134****5195</mobileCode>
      <userID></userID>
    </getMobileCodeInfo>
```

```
</soap:Body>
</soap:Envelope>
```

在线程组中新增监听器中的查看结果树，完善后的脚本如图 12-4-1 所示。

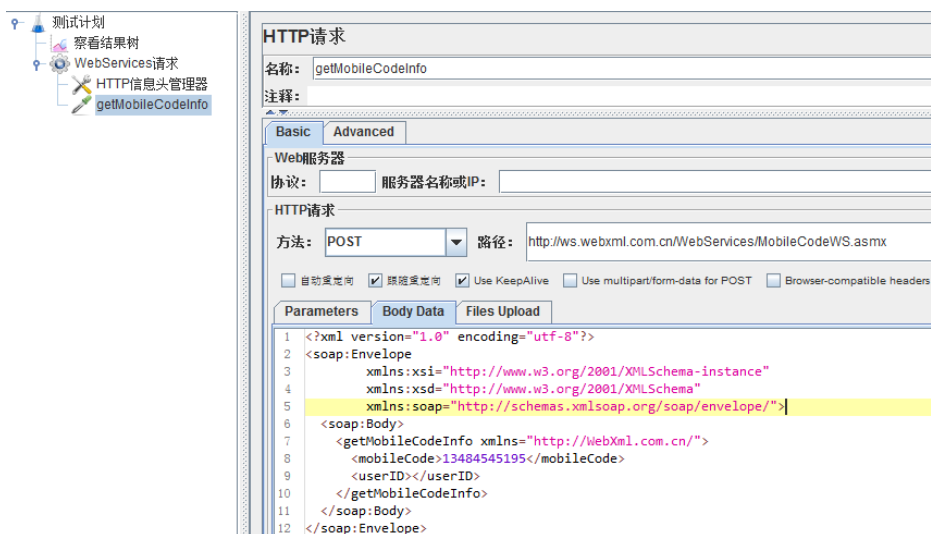



图 12-4-1

点击导航栏的 ，在结果树中可查看到服务端响应回复的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <getMobileCodeInfoResponse xmlns="http://WebXml.com.cn/">
      <getMobileCodeInfoResult>13484545187: 陕西 西安 陕西移动动感地带卡</getMobileCodeInfoResult>
    </getMobileCodeInfoResponse>
  </soap:Body>
</soap:Envelope>
```

12.5 HTTP 的请求

下面来看使用 JMeter 进行针对 HTTP 请求的接口测试。这里以“抽屉新热榜”产品为例。在登录中输入账号和错误的密码，然后点击“登录”按钮。打开

Chrome 浏览器调试功能的 Network 查看请求的信息，获取的请求信息为：

General:

```
Request URL:https://dig.chouti.com/login
Request Method:POST
Status Code:200
Remote Address:117.34.34.233:443
Referrer Policy:no-referrer-when-downgrade
```

Response Headers:

```
cache-control:no-cache
content-length:76
content-type:text/plain; charset=utf-8
date:Fri, 01 Feb 2019 07:16:29 GMT
eagleid:752222cf15490053896208101e
server:Tengine
status:200
timing-allow-origin: *
vary:Accept-Encoding
via:cache30.l2nu16-1[63,0], cache7.cn451[199,0]
```

Request Headers:

```
:authority:dig.chouti.com
:method:POST
:path:/login
:scheme:https
accept:/*/*
accept-encoding:gzip, deflate, br
accept-language:zh-CN,zh;q=0.9
content-length:49
content-type:application/x-www-form-urlencoded; charset=UTF-8
cookie:gpsd=698471c95aff035be8d4869235bc6225;
gpId=ac6249d0fb574589acc5def913eb8753; _9755xjdesxxd_=32;
YD00000980905869%3AWM_TID=IdnDYQ0lDnBARERRVY9kBbtC%2BfqlyID;
JSESSIONID=aaa0swU604MOfek-heeIw;
gdxidpyhxdE=fvIRMKot6%2Fqb73%2B60kGP8whPfMvT9WY9Dx4%5CWOH6zhcHb%5C4PJLcw
IghJe82KWlXmpKx0tUrZAudwfxd9dG7c%2F4oPfqhGbQy4lqEcjrMqxYW%2Brxnr1MkSBpmu
AldouulK8pQeBXsriLolAd779U6hQW8pEMSxwq2h9JwxJ1lQ%2Bitfqi2L%3A15490060556
76;YD00000980905869%3AWM_NI=sgSK4Xe2vULaUu4jLidE5YSyNag7zQiZ9zFmcQDWboE8
2G%2FPbv8Wg16f0Y3wO2xjlEOvwsG8%2FJxhpz3ezk5Mhtt%2FtFHsZFUXJWYqxzd45Aqgdm
6MV6Bfy6Th2h3KnnXzUUo%3D;YD00000980905869%3AWM_NIKE=9ca17ae2e6ffcdal70e2
e6ee8df867b7928ca6d95e819e8ba2d45b969b9f85ee7d8bf0bfcdd66282ebb696bb2af0
```

```
fea7c3b92ab1b99dacb754a7eebab6cf6b97abfc8bcf63a89bfbbad2738ebfa9d8f95bab
86f79bc567a18f8cb8e16587e8fea8ae5ca2aaf8b8e63bf6ba00bab27d8deeb88bb44f8d
aab9b4c634ed96b889f372f1bbb68cb847acacbcd0f74e8cad96a7d8548a899792fb72a8
ecfba2f46da8b3faaed53e82bab7d7d764f2b8fdaced669ca69bb6ee37e2a3
dnt:1
origin:https://dig.chouti.com
referer:https://dig.chouti.com/
user-agent:Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36
x-requested-with:XMLHttpRequest
```

Form Data 为:

```
phone:86134****5195
password:asdfghjkl
oneMonth:1
```

查看以上请求信息，可以得出如下结论：

请求方法：POST

请求 URL：https://dig.chouti.com/login

请求参数：

phone:86134****5195

password:asdfghjkl

oneMonth:1

请求 Headres 为：

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

确定如上信息后，在 JMeter 中再次创建一个新的线程组和新的 HTTP 请求，HTTP 信息头管理器如图 12-5-1 所示。

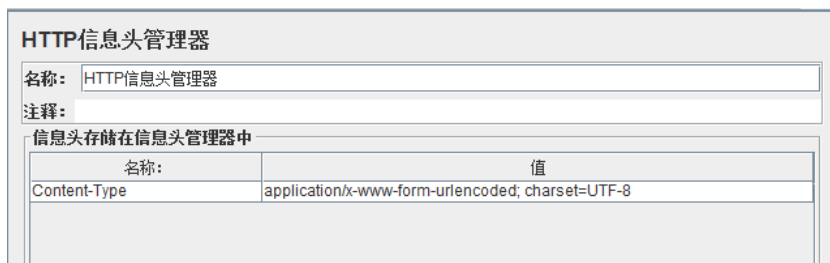


图 12-5-1

HTTP 请求的接口信息如图 12-5-2 所示。

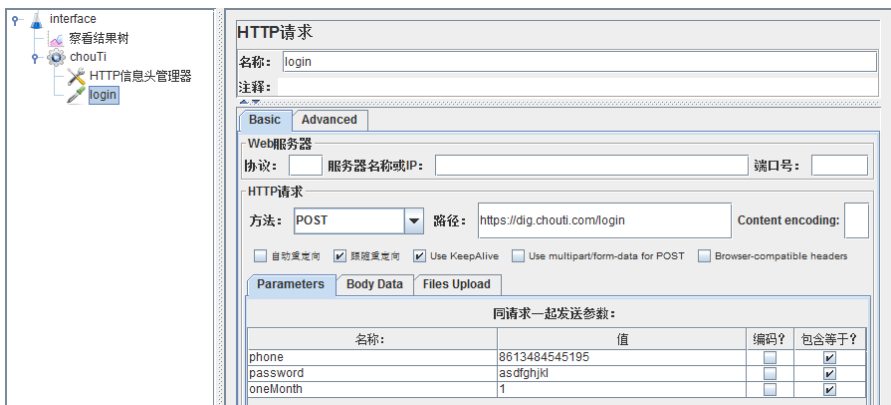


图 12-5-2

点击执行后，在“察看结果树”中可看到执行的结果如图 12-5-3 所示。

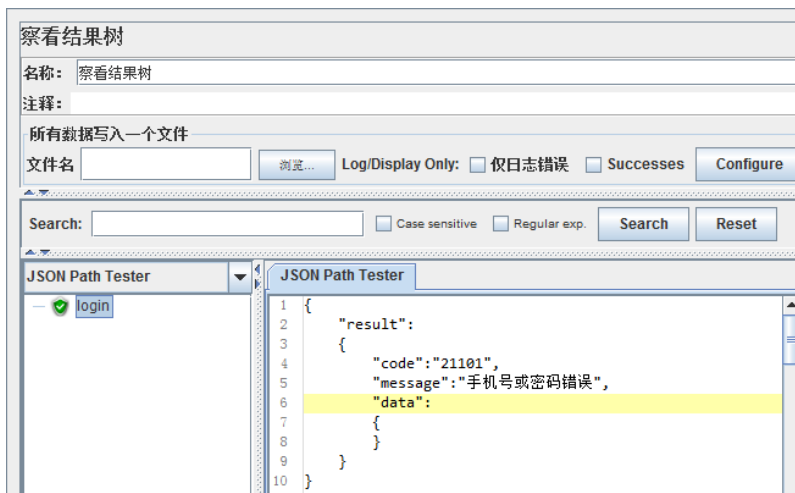


图 12-5-3

12.6 JMeter 断言实战

在接口测试中断言非常重要，一个没有断言的接口测试用例是无效的。用例中一个断言有三个层面，分别是 HTTP 状态码的断言：业务状态码的断言和某一接口请求后服务端响应数据的断言。在 JMeter 中增加断言的方式是，右键点击测试用例，在“添加”中选择“断言”。在断言中点击响应断言，在响应断言中，要测试的模式填写内容:HTTP/1.1 200 OK，要测试的响应字段选择 Response

Headers，模式匹配规则选择 Substring，把该响应断言命名为 HttpStatusCode，并且把该响应断言放在全局的位置，如图 12-6-1 所示。

响应断言

名称: HttpStatusCode

注释:

Apply to:

☐ Main sample and sub-samples ☒ Main sample only ☐ Sub-samples only ☐ JMeter Variable Name to use

要测试的响应字段

☐ 响应文本 ☐ 响应代码 ☐ 响应信息 ☒ Response Headers

☐ Request Headers ☐ URL样本 ☐ Document (text) ☐ Ignore Status

☐ Request Data

模式匹配规则

☐ 包括 ☐ 匹配 ☐ Equals ☒ Substring ☐ 否 ☐ 或者

要测试的模式

要测试的模式	
1	HTTP/1.1 200 OK

图 12-6-1

接下来添加业务状态码。在接口 login 中服务端响应数据中返回的业务状态码是 21101，用鼠标右键点击 login 接口用例，在断言中选择 JSON Assertion，在 Assert JSON Path exists 中编写获取到业务状态码的脚本，在 Expected Value 中编写期望的结果，完善后的内容如图 12-6-2 所示。

JSON Assertion

名称: JSON Assertion

注释:

Assert JSON Path exists: \$.result.code

☒ Additionally assert value

☒ Match as regular expression

Expected Value:

21101

☐ Expect null

☐ Invert assertion (will fail if above conditions met)

图 12-6-2

由于 login 接口用例无返回的 data 数据，所以就不做断言。对接口进行断言后，在监视器中新增断言结果，再次执行断言的结果如图 12-6-3 所示。

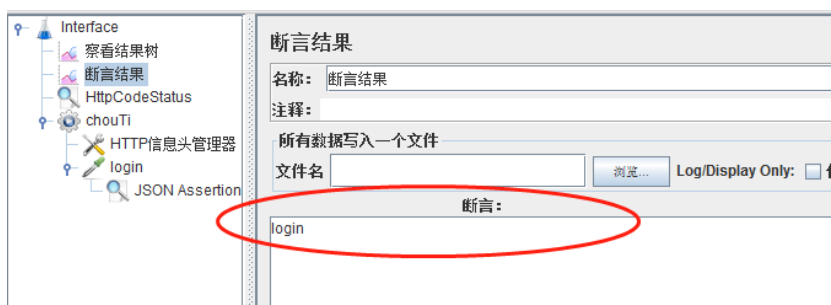


图 12-6-3

注解：如果断言失败，则会显示详细的失败信息。

12.7 HTTP 请求默认值

在 12.6 小节的实例代码中，login 的接口路径中直接填写着请求地址，但实际应用中，请求地址会发生改变，特别是在接口测试用例的情况下，如果请求地址或者端口发生了变化，每个接口用例都需要在路径中修改，这样维护的成本很高。在自动化测试思维体系中，无论是什么形式的自动化方式，维护方便都是必须要考虑的因素之一。是否可以把请求地址和端口单独分离出来放到一个地方，让所有接口用例都可以继承使用，这样，即使修改请求地址和端口，只需要维护一个地方即可，而无须修改每个接口测试用例。这里使用 JMeter 的 HTTP 请求默认值，它可以把请求地址和端口分离出来并被所有的接口用例应用。用鼠标右键点击线程组，在配置元件中选择 HTTP 请求默认值，在协议中填写 HTTPS，在服务器名称或者 IP 中填写请求的地址 dig.chouti.com，在端口号中默认为空，因为 HTTP 请求的默认值是针对所有接口用例的，所以位置移动到线程组下面，填写的内容和位置如图 12-7-1 所示。

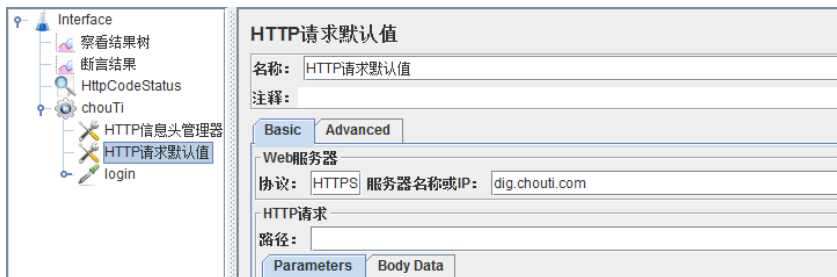


图 12-7-1

接下来修改接口用例。在接口用例的路径中不再需要填写请求地址和端口，只填写具体的接口名称，如原来 login 的路径为 https://dig.chouti.com/login，现在只需要填写/login 即可，修改后的内容如图 12-7-2 所示。

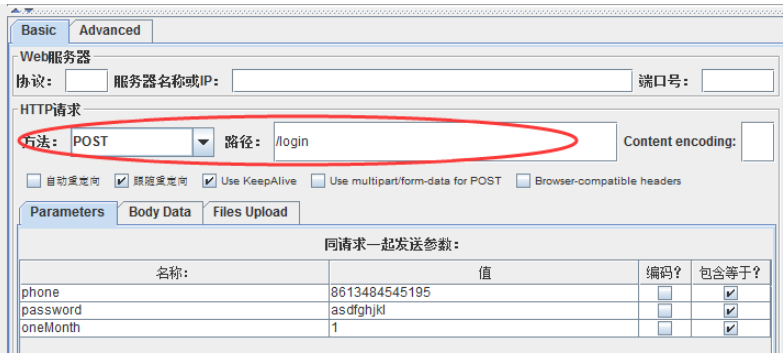


图 12-7-2

再次执行接口用例查看结果是否正确（之所以再次执行，是因为系统进行了修改，查看是否因为修改导致错误）。

12.8 用户定义的变量

为降低测试维护成本，通过以上 HTTP 请求默认值把请求地址等公共数据进行分离，公共数据主要涉及请求地址、登录系统的用户名和密码等数据，其他的数据可通过 JMeter 测试工具自己生成并使用生成之后的数据，使接口测试的数据形成一个闭环。在 JMeter 中新增用户定义的变量，新增的步骤为右键点击线程组，在配置元件中点击用户定义的变量，在用户定义的变量中新增公共数据请求地址、端口号、用户名和密码，用户定义的变量是全局的，移动到线程组下面，用户定义变量的公共数据和用户定义变量的位置如图 12-8-1 所示。



图 12-8-1

在图 12-8-1 中可以看到，在用户定义的变量中增加了 URL、PHONE 和 PASSWORD 变量，公共数据使用时只需要调用这几个变量。这里把请求地址放在了变量 URL 中。修改 HTTP 请求默认值，在服务器名称或 IP 中直接填写 URL 变量。在 JMeter 中调用变量的方式是 \${变量名}，在服务器名称或 IP 地址中填写 \${URL}，修改后如图 12-8-2 所示。

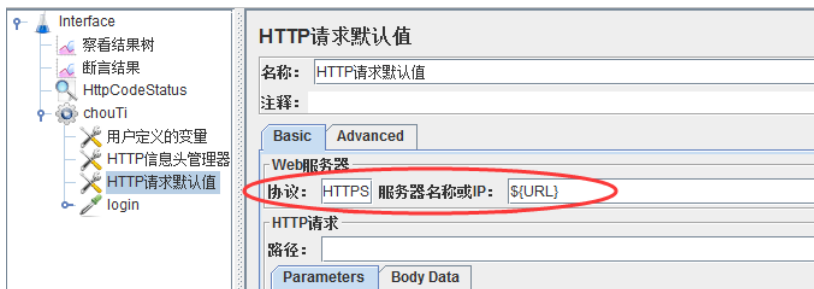


图 12-8-2

再次执行 login 的接口用例，执行结果通过，说明分离公共数据是正确的。
新增登录成功的接口用例，对用户名和密码使用调用变量的形式，登录成功的接口用例如图 12-8-3 所示。

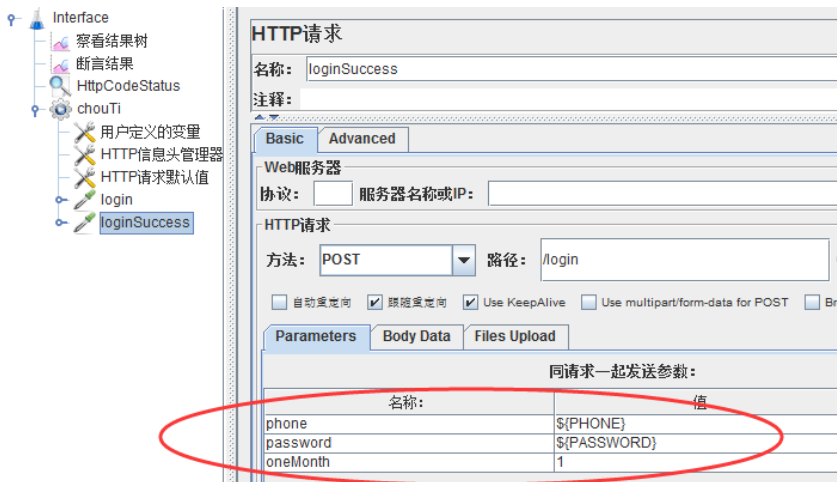


图 12-8-3

再次执行所有的用例，执行成功的信息，如图 12-8-4 所示。

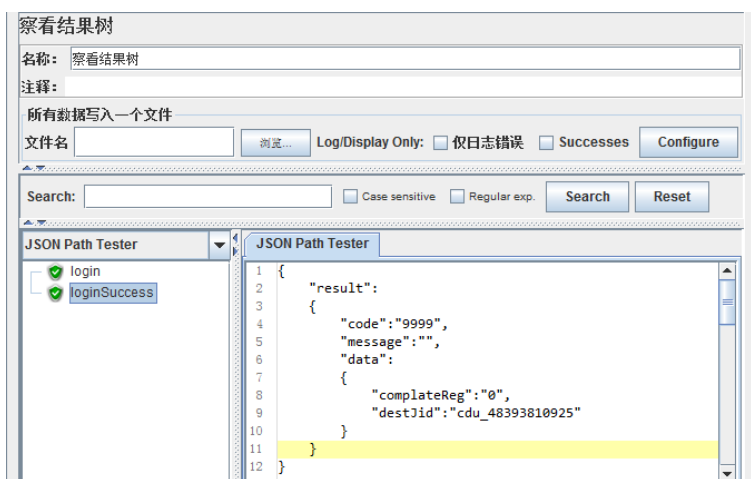


图 12-8-4

12.9 Token 的获取实战

在 PostMan 中介绍到登录系统成功后，服务端返回给客户端的响应数据中返回了 Token，在后面接口的请求中请求参数 Token 与登录成功后返回的 Token 一致。那么，在 JMeter 中如何获取这个 Token 呢？获取的方式有两种，一种方式是使用 JMeter 中后置处理器的正则表达式提取器获取，另外一种方式是使用后置处理器的 JSON Path Extractor 获取。

在 JMeter 的测试计划中创建“token 实战”的线程组，在该线程组中新增登录成功的接口用例，如图 12-9-1 所示。

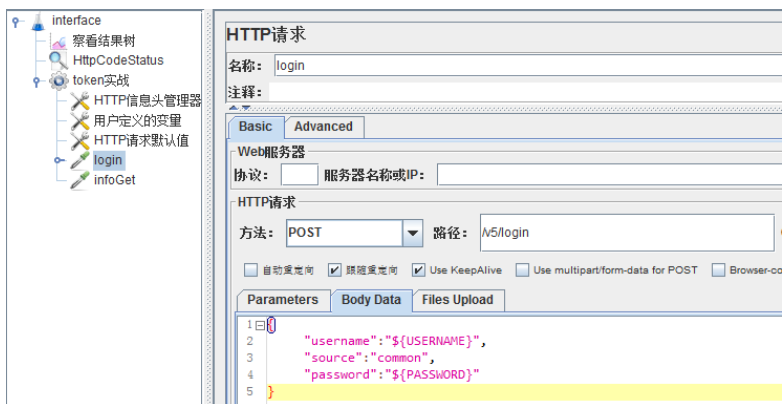


图 12-9-1

用鼠标右键点击 login 接口，在后置处理器中点击“正则表达式提取器”，填写获取 Token 的正则表达式，如图 12-9-2 所示。

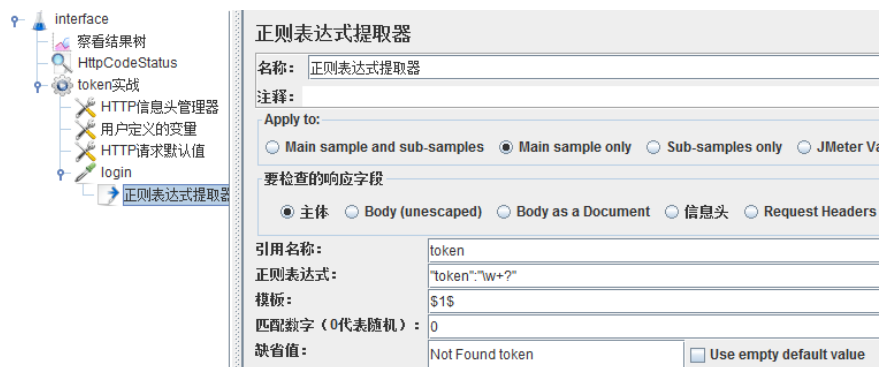


图 12-9-2

注解：这里变量名称为 token，模板和匹配数字默认，在正则表达式中匹配获取到登录成功后的 token 内容，默认编写 Not Found token，是指如果获取 token 失败，调用变量 token 的时候，会显示 Not Found Token 的信息。

在实际应用中，如需获取 Token 或者动态的参数，建议使用 JSON Path Extractor，这种获取方式更加简单。用鼠标右键点击 login 接口，在后置处理器中点击“JSON Path Extractor”，在 JSON Path Extractor 中填写获取 token 的内容，如图 12-9-3 所示。

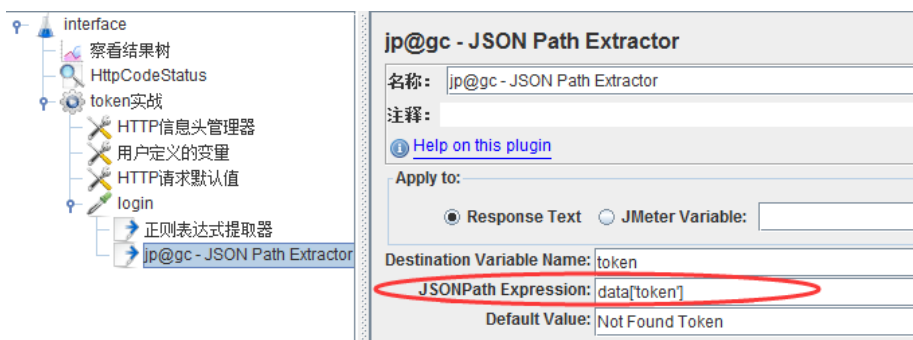


图 12-9-3

注解：在 Destination Variable Name 中定义要获取的变量名称 token，在 JSONPath Expression 中填写获取 token 的节点。由于 token 是在 data 下，所以填写 data['token']，在 Default Value 中填写当获取 token 失败，返回默认值信息。

这里使用了两种获取 token 的方式，如果确认使用 JSON Path Extractor 方法，那么就需要禁用正则表达式提取器，禁用的方式是在正则表达式提取器中用鼠标右键点击禁用，禁用后的效果如图 12-9-4 所示。

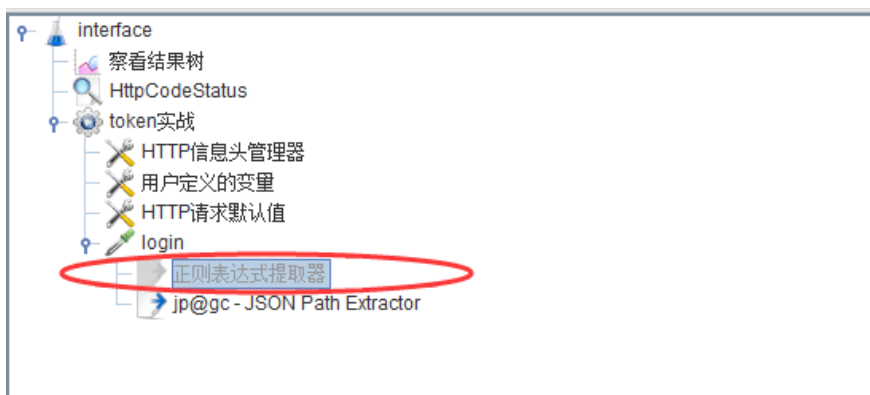


图 12-9-4

获取了登录成功后的 token 之后，下面再新增登录成功后的接口用例来应用定义的变量 token。新增接口 infoGet，它的请求参数是 {"token": "ma5TnNKAquu0dgX9vf41525330567364"}，请求参数中 token 是登录成功响应数据中的 token，在新增的 infoGet 接口中直接调用 token 的变量，infoGet 的接口内容如图 12-9-5 所示。

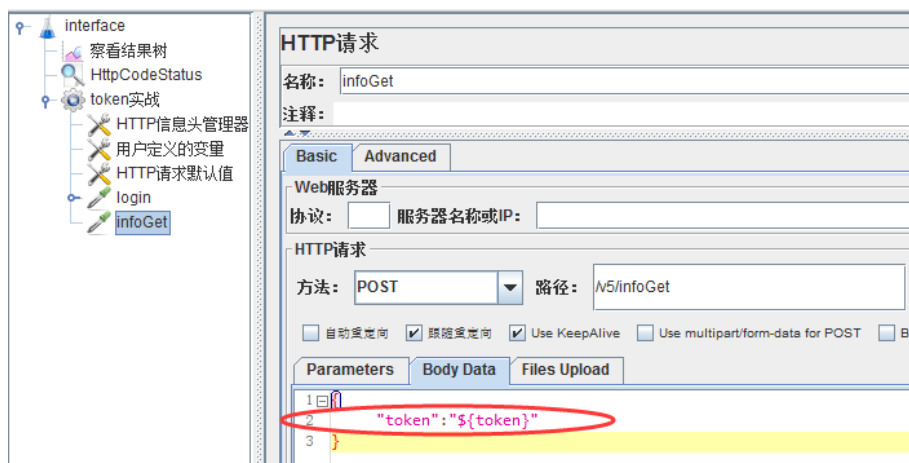


图 12-9-5

再次执行接口用例，infoGet 接口的请求参数如图 12-9-6 所示。

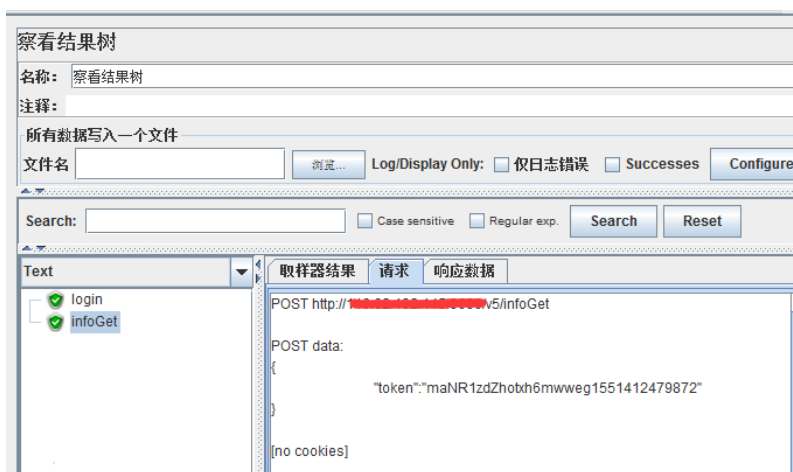


图 12-9-6

注解：在图 12-9-5 中可以看到，infoGet 的参数 token 获取 token 的变量成功。

12.10 HTTP Cookie 管理器实战

HTTP 是一个无状态的协议，在 12.9 小节中介绍过，登录成功后返回了 token，获取 token 等于拿到了令牌，用户就可以在系统执行相关业务。并不是所有的产品都采用 Token 签发令牌这样的方式，以人人网为例，来看 JMeter 测试工具中“HTTP Cookie 管理器”的应用。登录系统时用 Charles 抓包工具抓到的响应数据，如图 12-10-1 所示。

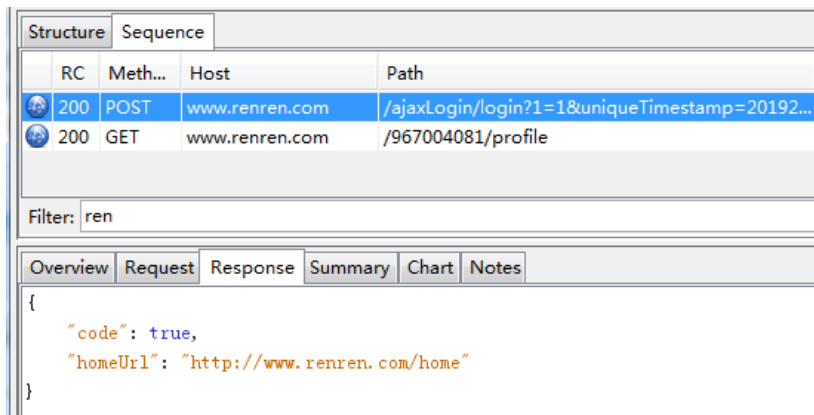


图 12-10-1

在图 12-10-1 中我们看到，登录成功后返回的响应数据中并没有 Token，查看个人主页请求地址 <http://www.renren.com/967004081/profile> 发送请求如何保证是在登录成功后的操作呢？查看个人主页 Request 的请求内容，如图 12-10-2 所示。

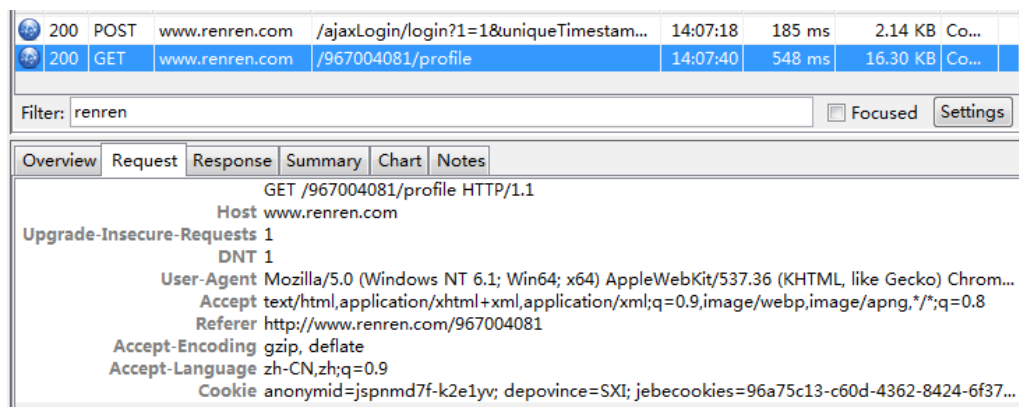


图 12-10-2

图中可以看到，个人主页地址 <http://www.renren.com/967004081/profile1> 在向服务端发送请求时带有 Cookie 信息，返回的内容是“无涯”个人主页的信息。在 JMeter 中创建线程组“人人网”，新增登录和个人主页的接口用例，如图 12-10-3 所示。



图 12-10-3

点击“启动”按钮执行接口用例后，个人主页接口用例返回的内容并不是“无涯”个人主页的内容，而是重定向到登录的内容，如图 12-10-4 所示。

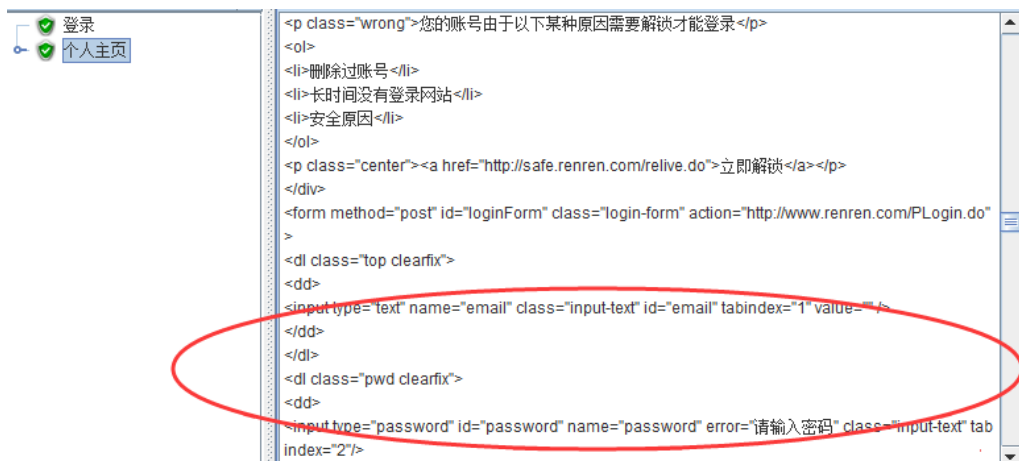


图 12-10-4

用鼠标右键点击线程组，在配置元件中点击“HTTP Cookie 管理器”，移动到查看结果树下面，如图 12-10-5 所示。



图 12-10-5

注解：添加 HTTP Cookie 管理器后，JMeter 会自动记录并保存服务端返回的 cookie 信息，并且在后面所有请求中自动添加 cookie，而且每个线程的 cookie 都是独立的。

执行以上接口用例，查看个人主页接口用例在请求时是否带了登录成功后的

标识，如图 12-10-6 所示。



图 12-10-6

个人主页返回的响应数据会显示“无涯”个人主页的信息，如图 12-10-7 所示。

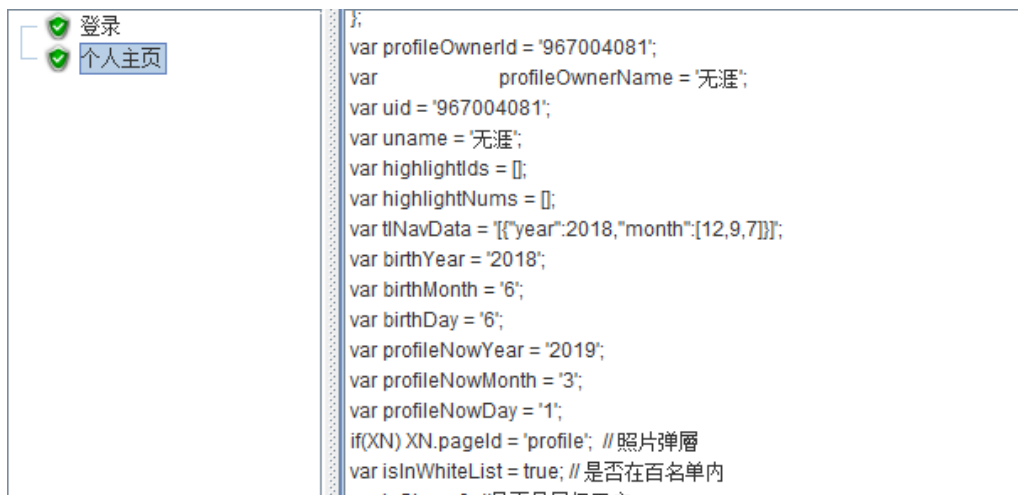
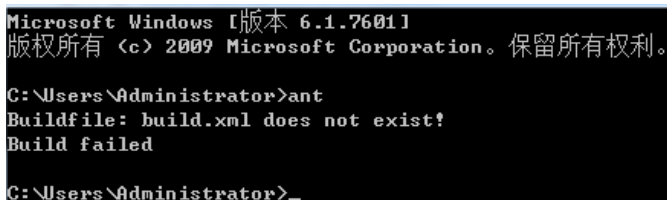


图 12-10-7

注解：在以上请求中可以看到，“个人主页”的接口用例在请求中自动带了登录成功后的标识，会返回用户个人主页的信息。

12.11 生成测试报告实战

虽然在 JMeter 的结果树中可以看到接口用例执行的结果，但是这样的结果看起来很不直观，下面，结合 Ant 工具生成基于 HTML 的测试报告。在 <https://ant.apache.org/bindownload.cgi> 下载 Ant，把 `apache-ant-1.10.1-bin.tar.gz` 下载后解压，把 Ant 所在的文件路径名填加到 `path` 的环境变量中，打开 `cmd` 命令提示符输入 `ant`，出现图 12-11-1 中显示的信息，表示 Ant 环境配置成功。



```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>ant
Buildfile: build.xml does not exist!
Build failed

C:\Users\Administrator>_
```

图 12-11-1

生成测试报告的步骤具体为：

(1) 修改 `bin` 目录下的 `jmeter.properties`，把 `jmeter.save.saveservice.output_format=csv` 修改为 `jmeter.save.saveservice.output_format=xml`，修改后的内容为：

```
# legitimate values: xml, csv, db. Only xml and csv are currently
supported.
#jmeter.save.saveservice.output_format=csv
jmeter.save.saveservice.output_format=xml
```

(2) 把 JMeter 的 `extras` 目录中的 `ant-jmeter-1.1.1.jar` 文件复制到 Ant 的 `lib` 目录下。

(3) 在 JMeter 的目录下创建 `testSuite` 目录，在 `testSuite` 目录下创建 `report` 目录和 `script` 的目录。`report` 目录用以存储生成的测试报告，在 `report` 目录下创建 `html` 目录，用以存储生成的基于 HTML 的测试报告，在 `report` 目录下创建 `jtl` 目录，用以存储生成的后缀为 `jtl` 的文件。`Script` 目录用以存储测试脚本（目前，测试脚本都在 `bin` 目录下，把测试脚本全部从 `bin` 命令中迁移到该目录下统一管理），在 `testSuite` 目录下创建 `build.xml` 文件，如图 12-11-2 所示。

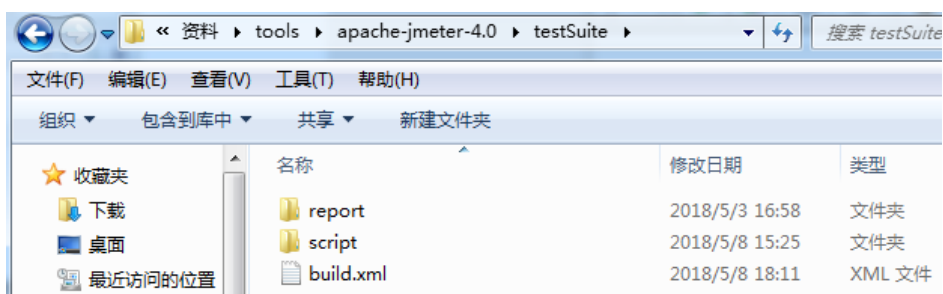


图 12-11-2

(4) 编写 build.xml 文件，Ant 执行的时候会用到该文件，build.xml 的内容为：

```
<?xml version="1.0" encoding="UTF8"?>

<project name="ant-jmeter-test" default="run" basedir=".">
<property name="jmeterPath" value="E:/桌面/资料/tools/apache-jmeter-4.0"/>

    <tstamp>
        <format property="time" pattern="yyyyMMddhhmm" />
    </tstamp>
    <property name="jmeter.home" value="{jmeterPath}" />
    <property name="jmeter.result.jtl.dir"
value="{jmeterPath}\testSuite\report\jtl" />
    <property name="jmeter.result.html.dir"
value="{jmeterPath}\testSuite\report\html" />
    <property name="htmlReportNameSummary" value="TestReport" />
    <property name="jmeter.result.jtlName"
value="{jmeter.result.jtl.dir}/{htmlReportNameSummary}${time}.jtl" />
    <property name="jmeter.result.htmlName"
value="{jmeter.result.html.dir}/{htmlReportNameSummary}${time}.html" />

    <target name="run">
        <antcall target="test" />
        <antcall target="report" />
    </target>

    <!--执行接口测试-->
    <target name="test">
        <echo>执行接口自动化测试</echo>
        <taskdef name="jmeter"
classname="org.programmerplanet.ant.taskdefs.jmeter.JMeterTask" />
```

```

        <jmeter jmeterhome="${jmeter.home}"
resultlog="${jmeter.result.jtlName}">
        <!--要执行的测试脚本-->
        <testplans dir="${jmeterPath}\testSuite\script"
includes="HTTP 请求.jmx" />
        <property name="jmeter.save.saveservice.output_format"
value="xml"/>
        </jmeter>
    </target>

    <!--解决报告中 NAN 字段显示问题-->
    <path id="xslt.classpath">
        <fileset dir="${jmeter.home}/lib" includes="xalan-2.7.2.jar"/>
        <fileset dir="${jmeter.home}/lib" includes="serializer-
2.7.2.jar"/>
    </path>

    <!--生成 HTML 测试报告-->
    <target name="report">
    <echo>生成接口自动测试报告</echo>
        <xslt classpathref="xslt.classpath"
force="true"
in="${jmeter.result.jtlName}"
out="${jmeter.result.htmlName}"
style="${jmeter.home}/extras/jmeter-results-detail-
report_21.xsl" />

        <!--复制图片-->
        <copy todir="${jmeter.result.html.dir}">
            <fileset dir="${jmeter.home}/extras">
                <include name="collapse.png" />
                <include name="expand.png" />
            </fileset>
        </copy>
    </target>

</project>

```

注解：在以上配置文件中，`jmeterPath` 指的是 `jmeter` 的目录（这个目录依据自己本地目录填写即可），`jmeter.result.jtl.dir` 指的是后缀为 `jtl` 的文件存储目录，`jmeter.result.html.dir` 指的是生成 `html` 测试报告的存储目录，`jmeter.result.jtlName` 指的是生成后缀为 `jtl` 文件的名称，`jmeter.result.htmlName` 指的是生成 `html` 测试报告的文件名称。在 `target` 为 `run` 中，先执行测试脚本，也就是 `<antcall target=`

"test" />, 执行测试脚本后生成测试报告<antcall target="report" />, 在执行测试脚本中需要指定测试脚本的具体目录, 也就是<testplans dir="{jmeterPath}\testSuite\script" includes="HTTP 请求.jmx" />, 在生成测试报告中需要指定测试报告模板以及复制 collapse.png 和 expand.png 图片。

(5) 到 build.xml 目录下执行 ant, 运行测试脚本和生成测试报告, 执行的过程如图 12-11-3 和图 12-11-4 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd E:\桌面\资料\tools\apache-jmeter-4.0\testSuite

C:\Users\Administrator>e:

E:\桌面\资料\tools\apache-jmeter-4.0\testSuite>ant
Buildfile: E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\build.xml

run:

test:
    [echo] 执行接口自动化测试
    [jmeter] Executing test plan: E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\script\HTTP请求.jmx ==> E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\jtl\TestReport201805030526.jtl
    [jmeter] Creating summariser <summary>
    [jmeter] Created the tree successfully using E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\script\HTTP请求.jmx
    [jmeter] Starting the test @ Thu May 03 17:26:22 CST 2018 (1525339582180)
```

图 12-11-3

```
[jmeter] Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
[jmeter] summary =      3 in 00:00:01 =    3.3/s Avg:   243 Min:   130 Max:
404 Err:      0 (0.00%)
[jmeter] Tidying up ...   @ Thu May 03 17:26:23 CST 2018 (1525339583492)
[jmeter] ... end of run

report:
    [echo] 生成接口自动测试报告
    [xslt] Processing E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\jtl\TestReport201805030526.jtl to E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\html\TestReport201805030526.html
    [xslt] Loading stylesheet E:\桌面\资料\tools\apache-jmeter-4.0\extras\jmeter-results-detail-report_21.xsl
    [copy] Copying 2 files to E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\html

BUILD SUCCESSFUL
Total time: 3 seconds
```

图 12-11-4

在图 12-11-3 和图 12-11-4 中可以看到执行成功, 在 report 目录下的 html 目录下存在 collapse.png, expand.png 和生成的 HTML 的测试报告。打开该 HTML

文件，基于 HTML 的测试报告内容如图 12-11-5 所示。

Load Test Results

Date report: date not defined

Designed for use with [JMeter](#) and [Ant](#).

Summary

# Samples	Failures	Success Rate	Average Time	Min Time	Max Time
3	0	100.00%	133 ms	79 ms	202 ms

Pages

URL	# Samples	Failures	Success Rate	Average Time	Min Time	Max Time	
login	1	0	100.00%	202 ms	202 ms	202 ms	+
loginSuccess	1	0	100.00%	119 ms	119 ms	119 ms	+
infoGet	1	0	100.00%	79 ms	79 ms	79 ms	+

图 12-11-5

注解：在图 12-11-5 的测试报告中，可以看到接口用例总数、失败数、成功率及执行的时间。

12.12 自动发送邮件实战

如果生成测试报告后自动发送给具体的人，用户体验就会非常好。接下来实现执行接口用例，生成测试报告并自动发送邮件。使用 Ant 发送邮件需要下载 mail.jar, activation.jar, commons-email-1.2.jar 文件，并且放在 Ant 目录下的 lib 子目录下。再次编辑 build.xml 文件，以实现执行完成后自动发送邮件的功能，编辑后的 build.xml 文件代码如下：

```
<?xml version="1.0" encoding="UTF8"?>

<project name="ant-jmeter-test" default="run" basedir=".">

  <property name="jmeterPath" value="E:/桌面/资料/tools/apache-jmeter-4.0"/>
  <property name="mailhost" value="smtp.sina.cn"/>
  <property name="username" value="wuyal303@sina.com"/>
  <property name="password" value="admi****"/>
  <property name="mailfrom" value="wuyal303@sina.com"/>
  <property name="mail_to" value="2839168630@qq.com"/>
  <property name="mailsubject" value="XX 系统接口自动化测试报告"/>
  <property name="mail_port" value="25"/>
  <property name="message" value="Hi! 请查收。这是 XX 系统接口自动化测试报告，如
```

有任何疑问，请联系我，谢谢！"/>

```
<tstamp>
  <format property="time" pattern="yyyyMMddhhmm" />
</tstamp>
<property name="jmeter.home" value="${jmeterPath}" />
<property name="jmeter.result.jtl.dir"
value="${jmeterPath}\testSuite\report\jtl" />
<property name="jmeter.result.html.dir"
value="${jmeterPath}\testSuite\report\html" />
<property name="htmlReportNameSummary" value="TestReport" />
<property name="jmeter.result.jtlName"
value="${jmeter.result.jtl.dir}/${htmlReportNameSummary}${time}.jtl" />
<property name="jmeter.result.htmlName"
value="${jmeter.result.html.dir}/${htmlReportNameSummary}${time}.html" />

<target name="run">
  <antcall target="test" />
  <antcall target="report" />
  <antcall target="sendEmail" />
</target>

<!--执行接口测试-->
<target name="test">
  <echo>执行接口自动化测试用例</echo>
  <taskdef name="jmeter"
classname="org.programmerplanet.ant.taskdefs.jmeter.JMeterTask" />
  <jmeter jmeterhome="${jmeter.home}"
resultlog="${jmeter.result.jtlName}">
    <!--要执行的测试脚本-->
    <testplans dir="${jmeterPath}\testSuite\script" includes="HTTP 请
求.jmx" />
    <property name="jmeter.save.saveservice.output_format"
value="xml"/>
  </jmeter>
</target>

<!--解决报告中 NAN 字段显示问题-->
<path id="xslt.classpath">
  <fileset dir="${jmeter.home}/lib" includes="xalan-2.7.2.jar"/>
  <fileset dir="${jmeter.home}/lib" includes="serializer-2.7.2.jar"/>
</path>

<!--生成 HTML 测试报告-->
<target name="report">
```

```

<echo>生成接口自动化测试报告</echo>
<xslt classpathref="xslt.classpath"
      force="true"
      in="${jmeter.result.jtlName}" out="${jmeter.result.htmlName}"
      style="${jmeter.home}/extras/jmeter-results-detail-
report_21.xsl" />

  <!--复制图片-->
  <copy todir="${jmeter.result.html.dir}">
    <fileset dir="${jmeter.home}/extras">
      <include name="collapse.png" />
      <include name="expand.png" />
    </fileset>
  </copy>
</target>

<!--自动发送邮件-->
<target name="sendEmail">
  <echo>发送自动化测试报告</echo>
  <mail mailhost="${mailhost}"
        ssl="ture"
        user="${username}"
        password="${password}"
        mailport="${mail_port}"
        subject="${mailsubject}"
        messagemimetype="text/html"
        tolist="${mail_to}">

    <from address="${mailfrom}" />

    <attachments>
      <!--${jmeter.home}/TestCase/report/html/-->
      <fileset dir="${jmeter.result.html.dir}">
        <include
name="${htmlReportNameSummary}${time}.html"/>
        <include name="collapse.png" />
        <include name="expand.png" />
      </fileset>
    </attachments>

    <message>
      ${message}
    </message>
  </mail>
</target>

</project>

```


注解：在自动发送邮件中，一定要填写正确的邮箱账号和密码，以及邮件服务器的端口和 smtp 服务器的地址。执行的顺序是先执行测试脚本，然后生成测试报告，最后自动发送邮件。

再次 ant 执行，执行的步骤如图 12-12-1 所示。

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd E:\桌面\资料\tools\apache-jmeter-4.0\testSuite

C:\Users\Administrator>e:

E:\桌面\资料\tools\apache-jmeter-4.0\testSuite>ant
Buildfile: E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\build.xml

run:

test:
    [echo] 执行接口自动化测试用例
    [jmeter] Executing test plan: E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\script\HTTP请求.jmx => E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\jtl\TestReport201805070539.jtl
    [jmeter] Creating summariser <summary>
    [jmeter] Created the tree successfully using E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\script\HTTP请求.jmx
    [jmeter] Starting the test @ Mon May 07 17:39:22 CST 2018 <1525685962241>
    [jmeter] Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
    [jmeter] summary =      3 in 00:00:01 =    5.8/s Avg:   117 Min:   74 Max:
    203 Err:      0 (0.00%)
    [jmeter] Tidying up ...      @ Mon May 07 17:39:23 CST 2018 <1525685963169>
    [jmeter] ... end of run

report:
    [echo] 生成接口自动化测试报告
    [xslt] Processing E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\jtl\TestReport201805070539.jtl to E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\report\html\TestReport201805070539.html
    [xslt] Loading stylesheet E:\桌面\资料\tools\apache-jmeter-4.0\extras\jmeter-results-detail-report_21.xsl

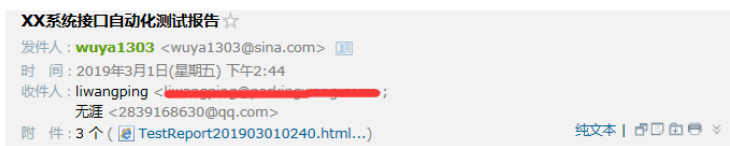
sendEmail:
    [echo] 发送自动化测试报告
    [mail] Sending email: XX系统接口自动化测试报告
    [mail] Sent email with 3 attachments

BUILD SUCCESSFUL
Total time: 14 seconds
```

图 12-12-1

注解：在图 12-12-1 中可以看到，执行的流程是先执行测试脚本，再生成基于 HTML 的测试报告，最后发送邮件给相关的人。

打开 QQ 邮箱 2938168630@qq.com，可以看到测试系统发送过来的邮件，如图 12-12-2 所示。



Hi! 请查收下, 这是XX系统接口自动化测试报告, 如有任何疑问, 请联系我, 谢谢!

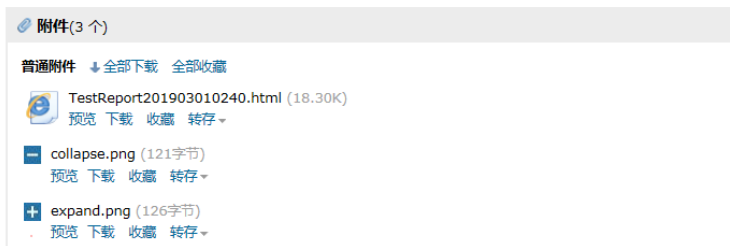


图 12-12-2

收件人可下载或者在线查看到系统的测试结果。

12.13 引入 CI

到 JMeter 的 testSuite 目录下执行 ant 命令, 可自动地执行接口用例和生成基于 HTML 的测试报告, 这些工作还可以通过 Jenkins 来完成, 直接在 CI 平台中选择“立即构建”选项, 就可以自动地完成, 而不需要每次到 testSuite 目录下执行 ant 命令, 同时生成的测试报告也可以在 Jenkins 平台中查看。

首先在 Jenkins 中配置 Ant 的路径, 点击“系统管理”按钮, 再点击“Global Tool Configuration”按钮, 在打开的页面中, 配置 Ant 的 ANT_HOME, 配置界面如图 12-13-1 所示。



图 12-13-1

注解：在 Name 中填写本地搭建的 ant 版本号，在 ANT_HOME 中填写 ant 在本地的路径，同时在插件中安装 ant。

在 Jenkins 中新建项目，名称为 Jmeter4.0，选择构建自由风格的软件项目，点击 OK 按钮后，在“增加构建”页面中选择“Invoke Ant”选项，点击“高级”按钮，在 Build File 中填写 build.xml 的路径，如图 12-13-2 所示。



图 12-13-2

增加构建后选择“Publish HTML reports”选项，点击“增加”按钮，在 HTML directory to archive 文本框中填写 HTML 的路径，在 Index page[s]中填写 HTML 的测试报告，填写内容如图 12-13-3 所示。

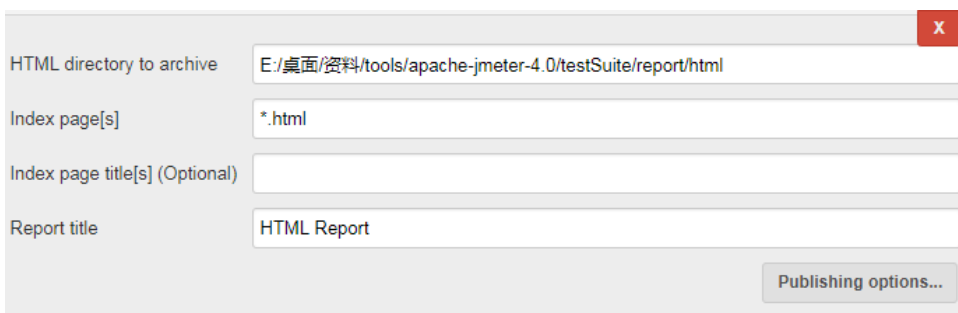


图 12-13-3

注解：使用正则表达式*.html，就可以获取所有基于 HTML 的测试报告。

点击“保存”按钮，跳转到项目 Jmeter 4.0 的详情界面，选择“立即构建”选项，构建的输出内容如下：

```
Started by user admin
Building in workspace
C:\Users\Administrator\.jenkins\workspace\Jmeter4.0
[testSuite] $ cmd.exe /C "ant.bat -file build.xml &&
exit %%ERRORLEVEL%%"
Buildfile: E:\桌面\资料\tools\apache-jmeter-4.0\testSuite\build.xml

run:

test:
    [echo] 执行接口自动化测试用例
    [jmeter] Executing test plan: E:\桌面\资料\tools\apache-jmeter-
4.0\testSuite\script\HTTP 请求.jmx ==> E:\桌面\资料\tools\apache-jmeter-
4.0\testSuite\report\jtl\TestReport201805081149.jtl
    [jmeter] Creating summariser <summary>
    [jmeter] Created the tree successfully using E:\桌面\资料
\tools\apache-jmeter-4.0\testSuite\script\HTTP 请求.jmx
    [jmeter] Starting the test @ Tue May 08 11:49:02 CST 2018
(1525751342906)
    [jmeter] Waiting for possible Shutdown/StopTestNow/Heapdump message
on port 4445
    [jmeter] summary +      1 in 00:00:00 =      2.4/s Avg:   202 Min:
202 Max:   202 Err:      0 (0.00%) Active: 1 Started: 1 Finished: 0
    [jmeter] summary +      2 in 00:00:00 =      9.7/s Avg:    99 Min:
79 Max:   119 Err:      0 (0.00%) Active: 0 Started: 1 Finished: 1
    [jmeter] summary =      3 in 00:00:01 =      4.8/s Avg:   133 Min:
79 Max:   202 Err:      0 (0.00%)
    [jmeter] Tidying up ...   @ Tue May 08 11:49:04 CST 2018
(1525751344378)
    [jmeter] ... end of run

report:
    [echo] 生成接口自动化测试报告
    [xslt] Processing E:\桌面\资料\tools\apache-jmeter-
4.0\testSuite\report\jtl\TestReport201805081149.jtl to E:\桌面\资料
\tools\apache-jmeter-
4.0\testSuite\report\html\TestReport201805081149.html
    [xslt] Loading stylesheet E:\桌面\资料\tools\apache-jmeter-
4.0\extras\jmeter-results-detail-report_21.xsl

sendEmail:
    [echo] 发送自动化测试报告
```

```
[mail] Sending email: XX 系统接口自动化测试报告
[mail] Sent email with 3 attachments

BUILD SUCCESSFUL
Total time: 21 seconds
[htmlpublisher] Archiving HTML reports...
[htmlpublisher] Archiving at PROJECT level E:/桌面/资料/tools/apache-jmeter-4.0/testSuite/report/html to
C:\Users\Administrator\.jenkins\jobs\Jmeter4.0\htmlreports\HTML_Report
Finished: SUCCESS
```

在项目 Jmeter 4.0 详情界面中可以看到生成的 HTML Report，如图 12-13-4 所示。



图 12-13-4

点击“HTML Report”按钮，会出现 HTML 测试报告界面，点击最新的测试报告，显示内容如图 12-13-5 所示。

Load Test Results

Date report: date not defined

Designed for use with [JMeter](#) and [Ant](#).

Summary

# Samples	Failures	Success Rate	Average Time	Min Time	Max Time
3	0	100.00%	567 ms	163 ms	1121 ms

图 12-13-5

12.14 JMeter 接口测试实战

前面详细地介绍了 JMeter 在接口测试中的应用，包括 cookie 的处理，token 的获取，JMeter 与 Ant 结合后测试报告的生成，自动发送邮件功能以及 Ant 与 Jenkins 的整合。本节将结合这些知识点使用 JMeter 测试工具进行一次接口测试的实战。

该业务内容要求首先成功登录系统，创建用户后查询用户、冻结用户（创建的用户默认状态是激活）、激活用户，最后删除用户。

在这样的一个业务中，接口测试主要包括三个维度，第一个是接口可用性的测试，主要用于验证一个接口请求是否正常，例如，登录的接口 login，执行成功只能说该接口请求正常，但是这还无法保证登录业务是成功的，因为在一个登录业务中不仅仅请求了 login 接口，还有其他的接口；第二个就是接口的校验，用于请求字段空值和边界值等校验；第三个就是通过接口测试技术来测试产品的业务。在接口用例执行成功后，接口用例执行结果全部通过，那么可以说明这个业务功能质量是合格的。本实例演示第三个维度，也就是通过接口测试来测试产品的业务。

启动 JMeter，在测试计划中创建新的线程组 userManage。在线程组创建用户定义的变量，HTTP 请求默认值和 HTTP 消息头管理器，在线程组中创建登录、创建用户、查询用户、冻结用户、激活用户和删除用户的简单控制器，如图 12-14-1 所示。

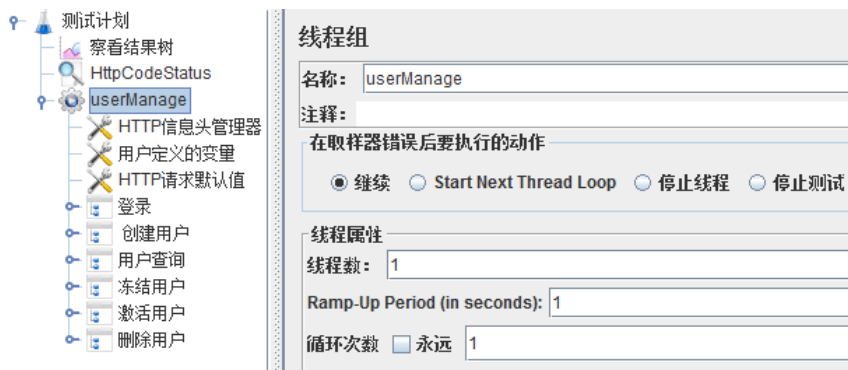


图 12-14-1

把请求地址、端口、用户名以及密码放在用户定义的变量中，如图 12-14-2

所示。

用户定义的变量

名称：用户定义的变量

注释：

用户定义的变量

名称：	值	Description
URL		
PORT		
USERNAME		
PASSWORD	8d969eef6ecad3c29a3a6292...	

图 12-14-2

在 HTTP 信息头管理器中填写客户端发送请求到服务端所带的 headers 信息，如图 12-14-3 所示。

HTTP信息头管理器

名称：HTTP信息头管理器

注释：

信息头存储在信息头管理器中

名称：	值
Content-Type	application/json; charset=UTF-8
Parkingwang-Client-Source	ParkingWangAPIClientWeb

图 12-14-3

在 HTTP 请求默认值中填写请求的地址和端口，直接调用用户定义的变量，如图 12-14-4 所示。

HTTP请求默认值

名称：HTTP请求默认值

注释：

BasicAdvanced

Web服务器

协议：服务器名称或IP：\${URL}端口号：

{PORT}

HTTP请求

路径：Content encoding：

ParametersBody Data

图 12-14-4

257

接下来完善登录业务的接口用例，主要有 login 和 infoGet，完善后的登录业务的接口如图 12-14-5 所示。

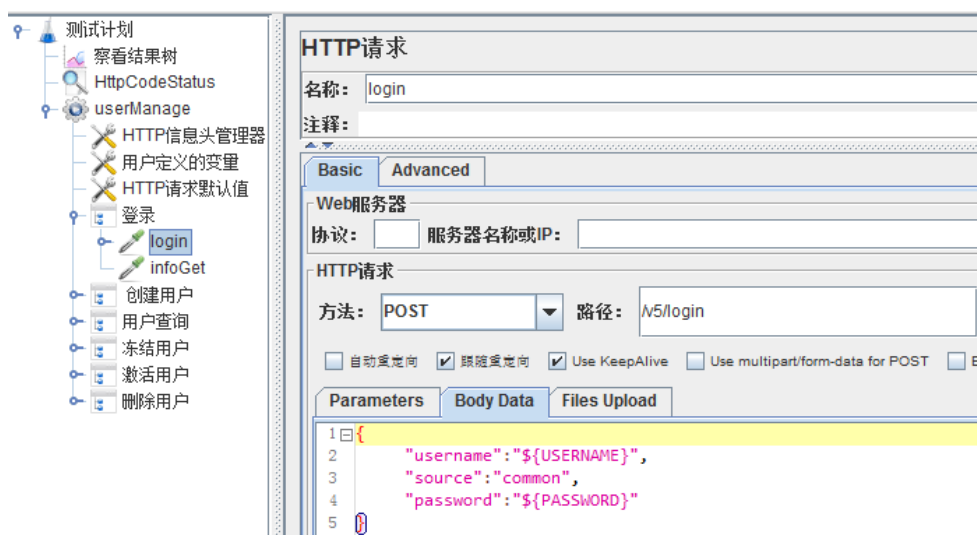


图 12-14-5

注解：在图 12-14-5 中，执行 login 接口成功后获取登录成功后的 token，以及对 login 进行断言验证 name 是否正确，校验 name 部分如图 12-14-6 所示。



图 12-14-6

接着执行 infoGet 接口用例。infoGet 接口用例中请求参数 token 与登录成功后返回的 token 必须一致，因此在 infoGet 请求参数中 token 值调用 login 接口用例后置处理器中定义的 token 变量，如图 12-14-7 所示。

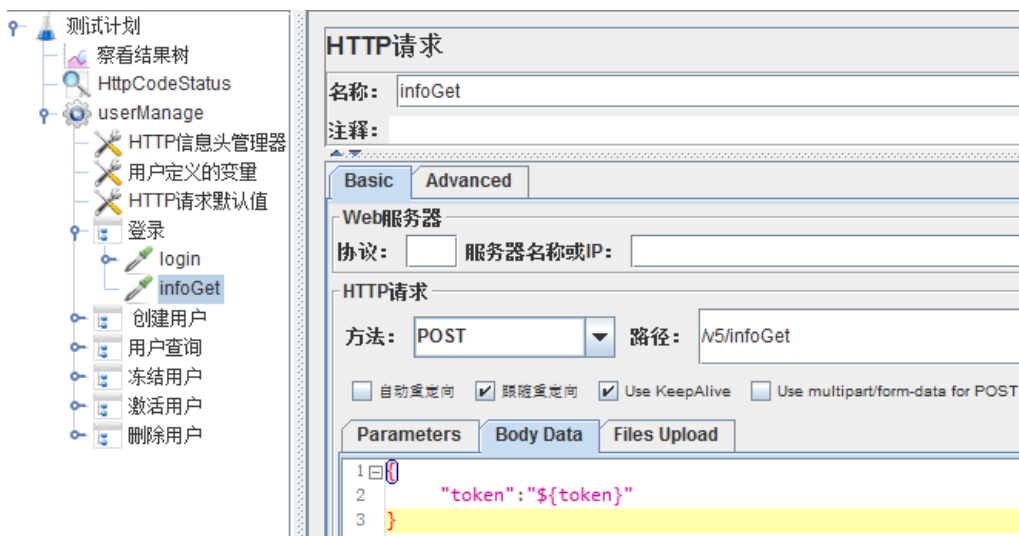


图 12-14-7

不管是 login 接口还是 infoGet 接口在请求成功后，响应数据中都包含了业务状态码，在线程组 userManager 中新增业务状态码断言，放在接口用例的上面对所有接口用例生效，业务状态码的断言如图 12-14-8 所示。



图 12-14-8

继续完善用户管理业务的接口用例，分别是创建用户，查询用户，冻结用户，激活用户和删除用户，如图 12-14-9 所示。



图 12-14-9

注解：在以上的用例中，可以看到用户业务执行的流程，即先创建用户然后对业务进行操作，这里特别说明，用户创建成功后默认状态是激活的，所以冻结用户用例应在激活用例前，而不能激活用例在冻结用例前。因为用户创建成功状态默认是激活的，再次执行激活的接口用例是有没意义的。在以上执行图中可以看到，其他接口都有断言验证，而冻结用户、激活用户及删除用户只验证了HTTP 协议状态码、业务状态码，但是没有断言响应数据。这样导致的问题是添加用户成功后，冻结了用户，用户的状态是否为冻结状态无法确定，但是在冻结用户、激活用户和删除用户中服务端返回的响应数据是{"status":0,"msg":"","data":{}}，那么如何证明冻结用户接口执行成功后用户的状态就是冻结的？可以调用查询用户接口来查看该用户的状态字段是否是冻结状态，如果是，证明冻结接口执行成功后用户确实已被冻结，激活用户同理。在冻结用户和激活用户用例后面添加查询用户接口用例，验证用户的状态。

冻结用户和激活用户后，增加查询用户的接口验证用户状态的用例，如图 12-14-10 所示。



图 12-14-10

在图 12-14-10 中可以看到，删除接口用例同样缺少数据断言。如何在接口用例中通过断言证明用户已被删除？可以在删除接口用例后，增加用户查询的接口用例，依据用户名称查看该用户是否存在，如果返回的数据是 0，那么该用户已删除，否则说明删除接口用例存在问题，完善后的接口用例如图 12-14-11 所示。

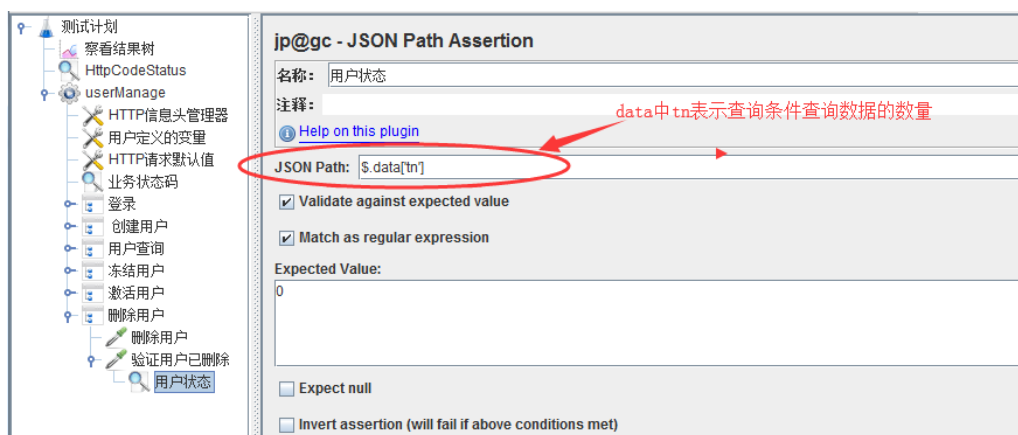


图 12-14-11

至此，用户业务的接口用例才比较完善了。用户业务如果使用手工的方式进行测试，至少需要 5 分钟以上的时间，而且每次环境更新，每次迭代这个业务都

需要重新测试。通过使用 JMeter 测试工具来实现该业务的接口用例，只需要几秒钟时间，就可以测试出该业务功能是否正常。为了确保接口 XX 模块的业务接口用例是正常、合理的，必须做到断言合理，也就是说断言应包含 HTTP 协议状态码验证、业务状态码和业务数据验证。另外还需要确保接口用例执行流程业务合理，例如，冻结用户用例时，需要明确地告诉我们冻结用户功能是否存在问题，也就是必须验证冻结用户后用户的状态，这里添加了查询用户的接口用例来验证用户的状态。假设冻结用户后没有执行用户查询接口和验证用户状态，是无法确定用户状态是否为冻结的。在自动化测试中，答案只有两种，一是成功，表明业务正确；二是失败，表明业务存在问题。

在 build.xml 文件中将 HTTP 请求.jmx 修改为 shop.jmx，在 Jenkins 中执行 Jmeter 4.0 的项目，执行后的结果如图 12-14-12 所示。

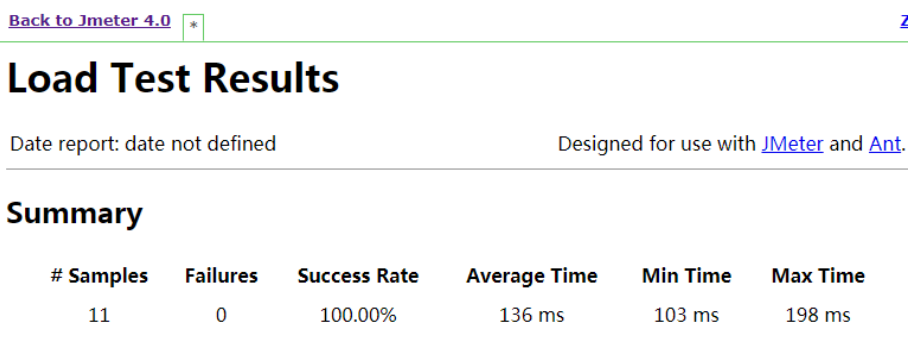


图 12-14-12

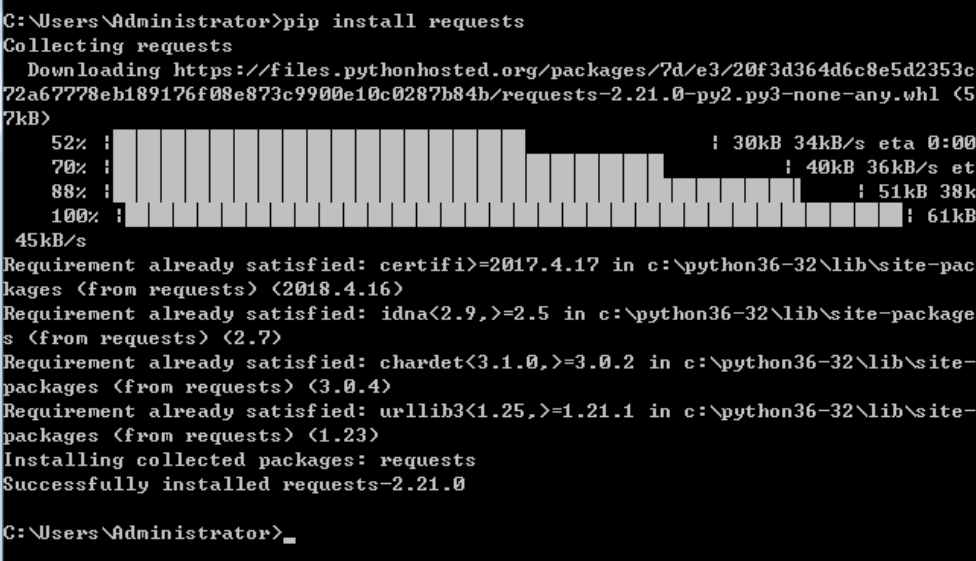
第 13 章 Requests 实战

13.1 Requests 简述

Requests 是 Python 语言的第三方库，专门用于发送 HTTP 请求。在 Python 语言中虽然提供了 urllib2 和 urllib 的库，但是相比较而言，Requests 仍然是实现接口测试的最好选择，因为它使用起来更加简便。由于它是第三方库，首先需要安装它，安装 Requests 的命令为：

```
pip install requests
```

安装该库的界面如图 13-1-1 所示。



```
C:\Users\Administrator>pip install requests
Collecting requests
  Downloading https://files.pythonhosted.org/packages/7d/e3/20f3d364d6c8e5d2353c72a67778eb189176f08e873c9900e10c0287b84b/requests-2.21.0-py2.py3-none-any.whl (57kB)
    52% |#####| 30kB 34kB/s eta 0:00
    70% |#####| 40kB 36kB/s et
    88% |#####| 51kB 38k
   100% |#####| 61kB
45kB/s
Requirement already satisfied: certifi>=2017.4.17 in c:\python36-32\lib\site-packages (from requests) (2018.4.16)
Requirement already satisfied: idna<2.9,>=2.5 in c:\python36-32\lib\site-packages (from requests) (2.7)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\python36-32\lib\site-packages (from requests) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in c:\python36-32\lib\site-packages (from requests) (1.23)
Installing collected packages: requests
Successfully installed requests-2.21.0

C:\Users\Administrator>_
```

图 13-1-1

13.2 Requests 发送请求

在 Requests 中，可以对常用的 HTTP 请求方法发送请求，在 request 函数中，它的第一个参数是 method，也就是 HTTP 的请求方法；第二个参数是 URL 也就是请求地址；第三个请求参数包含了 cookies, params 等。这里以请求淘宝某一个接口地址为例，来学习 Requests 库的应用。

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

r=requests.get('https://cart.taobao.com/trail_mini_cart.htm')
print(r.text)
```

以上代码执行的过程，实际上就是客户端发送请求到服务端，服务端返回响应数据，响应数据的内容如下：

```
(
  {
    "status" : false,
    "period": "3",
    "isLogin": "false",
    "promotionCount" : 0,
    "quailtyTenseCount" : 0,
    "item" : [
      ],
    "num" : -2
    , "errMsg" : "亲，重新登录试试！"
  }
)
```

在以上请求中，r 是 Response 对象，依据 r 可以查看服务端返回的 HTTP 状态码，以及 headers 等信息，修改后的代码为：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya
```

```
import requests

r=requests.get('https://cart.taobao.com/trail_mini_cart.htm')
print('HTTP 协议返回的状态码: \n{0}'.format(r.status_code))
print('返回的 Headers 信息: \n{0}'.format(r.headers))
print('返回的 cookies 信息为: \n{0}'.format(r.cookies))
print('返回的响应数据内容: \n{0}'.format(r.text))
```

在以上代码中，客户端发送请求后，服务端返回对应的状态码，headers 信息，cookie 和响应内容，执行后返回的内容如下：

HTTP 协议返回的状态码：
200

返回的 Headers 信息：

```
{'Server': 'Tengine', 'Content-Type': 'text/html;charset=GBK',
'Transfer-Encoding': 'chunked', 'Connection': 'keep-alive', 'Date': 'Thu,
17 May 2018 10:42:48 GMT', 'ufe-result': 'A6', 'Vary': 'Accept-Encoding',
'S': 'STATUS_NOT_EXISTED', 'P3P': 'CP='CURa ADMa DEVa PSAo PSDo OUR BUS
UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR"', 'Set-Cookie':
'ucl=cookie14=UoTeOLwBaEBnYg%3D%3D; Domain=.taobao.com; Path=/,
t=2874faea4213ce7811dbb2ff5ecb15a3; Domain=.taobao.com; Expires=Wed, 15-
Aug-2018 10:42:48 GMT; Path=/,
cookie2=3429087a5d988c16be4555f349994d52; Domain=.taobao.com; Path=/; HttpO
nly, v=0; Domain=.taobao.com; Path=/, _tb_token_=e573760feee5;
Domain=.taobao.com; Path=/, thw=cn; Path=; Domain=.taobao.com;
Expires=Fri, 17-May-19 10:42:48 GMT;', 'Content-Language': 'zh-CN',
'Expires': 'Thu, 17 May 2018 10:42:48 GMT', 'Cache-Control': 'max-age=0',
'Content-Encoding': 'gzip', 'Strict-Transport-Security': 'max-
age=31536000', 'Timing-Allow-Origin': '*', '*', 'EagleEye-TraceId':
'015101d115265537685372316e', 'Via': 'cache9.cn800[186,0]', 'EagleId':
'015101d115265537685372316e'}
```

返回的 cookies 信息为：

```
<RequestsCookieJar[<Cookie _tb_token_=e573760feee5 for .taobao.com/>,
<Cookie cookie2=3429087a5d988c16be4555f349994d52 for .taobao.com/>,
<Cookie t=2874faea4213ce7811dbb2ff5ecb15a3 for .taobao.com/>, <Cookie
thw=cn for .taobao.com/>, <Cookie ucl=cookie14=UoTeOLwBaEBnYg%3D%3D
for .taobao.com/>, <Cookie v=0 for .taobao.com/>]>
```

返回的响应数据内容：

```
(
{
    "status" : false,
```

```

        "period": "3",
        "isLogin": "false",
        "promotionCount" : 0,
        "quailtyTenseCount" : 0,
        "item" : [
                                ],
        "num" : -2
        , "errMsg" : "亲，重新登录试试！"
    }
)

```

13.3 URL 参数实战

在淘宝接口请求中，请求的 URL 为 `https://cart.taobao.com/trail_mini_cart.htm?callback=MiniCart.setData&t=1526048972328`，请求方法是 GET。那么如何在请求中处理 URL 的参数呢？在发送 GET 请求的时候，对接口后面如 `callback=MiniCart.setData&t=1526048972328` 参数进行处理，将其变成字典类型传递到 `params` 中，也就是说 `params` 参数是字典数据类型，对应的 `key` 值指定对应的 `value` 值，实现的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

r=requests.get(url='https://cart.taobao.com/trail_mini_cart.htm',

params={'callback':'MiniCart.setData','t':'1526048972328'})

print('请求的 URL 为: \n{0}'.format(r.url))

```

在以上代码中，发送 GET 请求时，在 `params` 参数中指定 URL 参数中 `key` 值对应的 `value` 值，运行以上代码后输出的结果为：

```

请求的 URL 为:
https://cart.taobao.com/trail_mini_cart.htm?callback=MiniCart.setData&t=1526048972328

```


13.4 请求头的添加

在客户端向服务端发送请求时需要带上请求头，也就是 Request Headers，服务端响应回复会带上响应头，也就是 Response Headers。在淘宝的接口请求中，如果需要在请求中带上请求头，用到的参数是 headers，也就是说在 headers 参数中指定请求头，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

r=requests.get(
url='https://cart.taobao.com/trail_mini_cart.htm',

params={'callback':'MiniCart.setData','t':'1526048972328'},
        headers={
'User-Agent':'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36',
        'Content-Type':'application/json',
'Referer':'https://shoucang.taobao.com/shop_collect_list.htm?spm=a21bo.2
017.1997525053.3.5af911d9sYX701'})

print('响应内容: \n{0}'.format(r.text))
```

以上代码在请求的时候带了请求头，即在参数 headers 中带了 Content-Type、User-Agent 和 Referer，运行以上代码后，服务端会返回响应数据，响应数据的内容为：

```
MiniCart.setData(
{
    "status" : false,
    "period": "3",
    "isLogin": "false",
    "promotionCount" : 0,
    "quailtyTenseCount" : 0,
    "item" : [
        ],
    "num" : -2
```

```

        , "errMsg" : "亲, 重新登录试试! "
    }
)

```

13.5 data 参数实战

在发送 POST 请求时会用到 `data` 的参数，它的功能是在客户端发送请求时将字典类型的值传进去，也就是说 `data` 参数值的数据类型是字典，如图 13-5-1 所示。

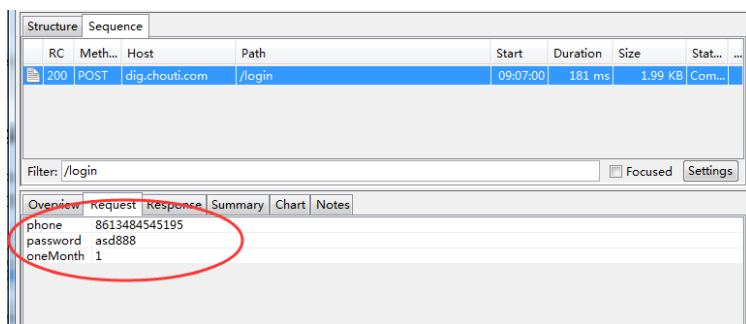


图 13-5-1

在图 13-5-1 中可以看到请求的参数是字典类型的数据类型，请求方法是 POST。下面请求该接口，参数使用 `data`，实现的代码如下：

```

#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import json
import urllib3
urllib3.disable_warnings()

def getHeaders():
    headers = {
        'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8',
        'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81
Safari/537.36'
    }
    return headers

dict1={'phone':'86134****5195','password':'asd***','oneMonth':1}

```

```
def chouTi():
    r=requests.post(
        url='https://dig.chouti.com/login',
        data=dict1,
        headers=getHeaders(),
        verify=False)
    print(json.dumps(r.json(),indent=True))

if __name__ == '__main__':
    chouTi()
```

注释：在以上代码中，因为请求地址是 HTTPS，所以在请求中加了忽略安全证书的参数 `verify`，把它设置为 `False`。但是即使是这样，代码执行后依然会报 `C:\Python36-32\lib\site-packages\urllib3\connectionpool.py:857: InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings InsecureRequestWarning` 的警告信息，这是因为 `requests` 库是基于 `urllib` 封装的，在 `urllib3` 中，官方强制验证 HTTPS 的安全证书，如果没有是不能通过验证的。虽然在代码中添加了忽略警告的参数 `verify`，但是依然会显示醒目的警告信息。解决办法是添加如下两行代码：

```
import urllib3
urllib3.disable_warnings()
```

在实例代码中，可以看到发送 POST 请求的时候，请求参数 `data` 后面的参数 `dict1` 实际它的数据类型是字典（dict）。运行以上代码后，返回的结果信息如图 13-5-2 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-5.py
{
  "result": {
    "code": "9999",
    "message": "",
    "data": {
      "complateReg": "0",
      "destJid": "cdu_48393810925"
    }
  }
}

Process finished with exit code 0
```

图 13-5-2

13.6 JSON 参数实战

在 POST 请求中另一个经常用到的参数是 json，主要用于发送 JSON 可序列化的 Python 对象。如果请求头中 Content-Type 对应的 value 值是 application/json，并且发送请求的参数是 data 就需要对数据进行序列化处理，而如果直接使用 json 参数，就不需要对数据进行序列化处理了。这里结合具体的实例来说明这部分的应用，如图 13-6-1 所示。

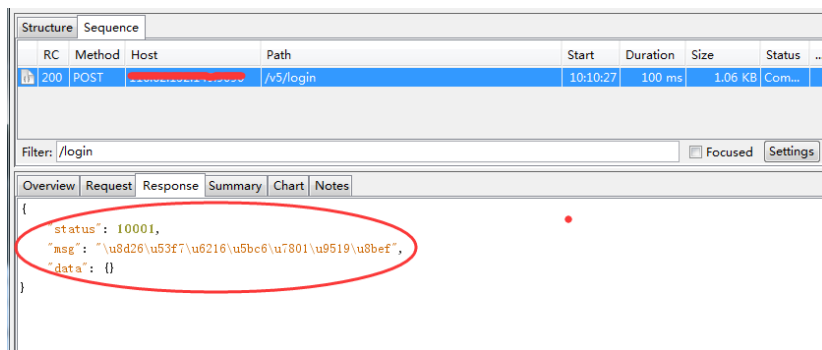


图 13-6-1

在图 13-6-1 中可以看到，客户端发送请求后，服务端返回的响应数据中的业务状态码 status 是 10001。在这里先以发送 POST 请求参数 data 为例，那么就需要对请求的参数做序列化的处理，实现的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import json

def getHeaders():
    headers={
        'Content-Type':'application/json;charset=UTF-8',
        'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'
    }
    return headers

dict1={"source":"common","password":""}
```

```
def login():
    r=requests.post(
        url='http://118.***.***.145:9999/v5/login',
        data=json.dumps(dict1),
        headers=getHeaders())
    print(json.dumps(r.json(),indent=True,ensure_ascii=False))

if __name__ == '__main__':
    login()
```

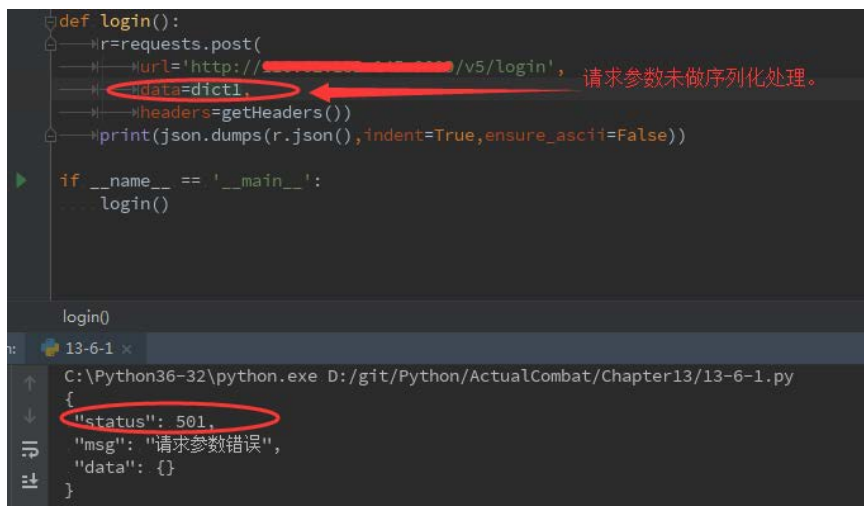
执行以上代码后，显示结果如图 13-6-2 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-6-1.py
{
  "status": 10001,
  "msg": "账号或密码错误",
  "data": {}
}

Process finished with exit code 0
```

图 13-6-2

在该实例中，如果发送的 POST 请求参数 data 不做序列化的处理，也就是 data=dict1，运行代码后，返回的错误信息并不是业务状态码 10001，而是 501 请求参数错误，如图 13-6-3 所示。



```
def login():
    r=requests.post(
        url='http://118.***.***.145:9999/v5/login',
        data=dict1,
        headers=getHeaders())
    print(json.dumps(r.json(),indent=True,ensure_ascii=False))

if __name__ == '__main__':
    login()
```

```
{
  "status": 501,
  "msg": "请求参数错误",
  "data": {}
}
```

图 13-6-3

在以上代码中，对请求数据做序列化的处理相对来说有点麻烦。而使用请求参数 json，就不需要对请求的数据做序列化的处理，修改后的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import json

def getHeaders():
    headers={
        'Content-Type': 'application/json;charset=UTF-8',
        'Parkingwang-Client-Source': 'ParkingWangAPIClientWeb'
    }
    return headers

dict1={"source": "common", "password": ""}

def login():
    r=requests.post(
        url='http://118.***.***.145:9999/v5/login',
        json=dict1,
        headers=getHeaders())
    print(json.dumps(r.json(), indent=True, ensure_ascii=False))

if __name__ == '__main__':
    login()
```

再次运行代码，可以看到返回的响应数据，如图 13-6-4 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-6-1.py
{
  "status": 10001,
  "msg": "账号或密码错误",
  "data": {}
}

Process finished with exit code 0
```

图 13-6-4

13.7 Token 实战

在前面 PostMan 和 JMeter 测试工具详细介绍了 Token，以及如何在 JMeter

和 PostMan 测试工具中获取和调用 Token。下面通过一个具体的实例，进一步理解 Token 的应用。如图 13-7-1 所示。

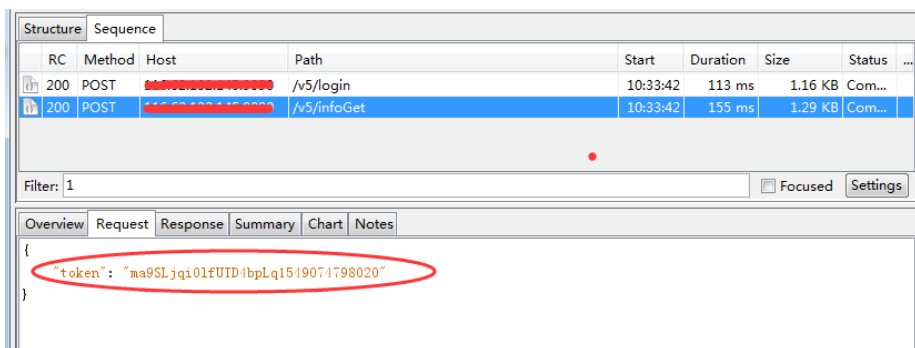


图 13-7-1

在图 13-7-1 中，可以看到登录成功后返回的 token，再次请求时需要带上参数 token，这个 token 和登录返回的 token 一致，实现的思路是：

- (1) 请求 login 接口登录成功；
- (2) 在响应数据中获取 token 并且返回 token；
- (3) infoGet 接口发送请求的时候带上 login 接口返回的 token。

实现的代码如下：

```

#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import json

def getHeaders():
    headers={
        'Content-Type': 'application/json;charset=UTF-8',
        'Parkingwang-Client-Source': 'ParkingWangAPIClientWeb'
    }
    return headers

def getUrl():
    return 'http://116.***.***.145:***'

dict1={"username": "66**66", "source": "common", "password":
"8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92"}
    
```

```

def login():
    r=requests.post(
        url=getUrl()+'/v5/login',
        json=dict1,
        headers=getHeaders())
    return r.json()['data']['token']

def infoGet():
    r=requests.post(
        url=getUrl()+'/v5/infoGet',
        json={'token':login()},
        headers=getHeaders())
    return r.json()

if __name__ == '__main__':
    infoGet()

```

执行 infoGet 函数后，会打印出用户的详细信息。这种方法的缺点是每次调用登录后的接口，都需先调用 login 函数，非常不方便。现在，我们可以把登录成功后返回的 Token 写入到文件中，后面测试其他接口时直接读取文件里面的 Token 值即可。下面结合 unittest 单元测试框架来修改代码，修改后的代码如下：

```

#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import unittest

def getHeaders():
    headers={
        'Content-Type':'application/json;charset=UTF-8',
        'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'}
    return headers

def getUrl():
    return 'http://116.***.***.145:****'

def loginData():
    dict1 = {"username": "66**66", "source": "common",
        "password":
        "8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92"}

```



```

        return dict1

class Api(unittest.TestCase):
    def statusCode(self,r):
        self.assertEqual(r.json()['status'],0)
        self.assertEqual(r.status_code,200)

    def test_api_001(self):
        ''' 登录系统'''
        r=requests.post(
            url=getUrl()+'/v5/login',
            json=loginData(),headers=getHeaders())
        with open('token','w') as f:
            f.write(r.json()['data']['token'])

    def getToken(self):
        ''' 读取 token 文件内容'''
        with open('token','r') as f:
            return f.read()

    def test_api_002(self):
        ''' 获取用户信息'''
        r=requests.post(
            url=getUrl()+'/v5/infoGet',
            json={'token':self.getToken()},
            headers=getHeaders())
        self.statusCode(r=r)
if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在以上代码中，把登录的代码放在了 `test_api_001` 测试用例中，然后把登录成功后返回的 `token` 写入到文件 `token` 中，通过 `getToken` 方法从文件中读取存储 `token` 的文件，以后其他接口再需要使用 `Token` 的时候，直接调用 `getToken` 方法即可。`getHeaders` 是为 `headers` 参数单独编写的函数，在需要用到的时候直接调用这个方法即可；返回请求地址的函数是 `getUrl`。自动化测试用例，必须要有断言，而接口用例的断言需要验证 `HTTP` 的状态码、业务状态码和具体接口的数据。不管哪个接口用例，都必须验证 `HTTP` 的状态码和业务状态码，将这部分内容单独写一个方法 `statusCode`，每个接口用例的断言直接调用这个方法即可。执行以上代码，执行后的结果如图 13-7-2 所示。

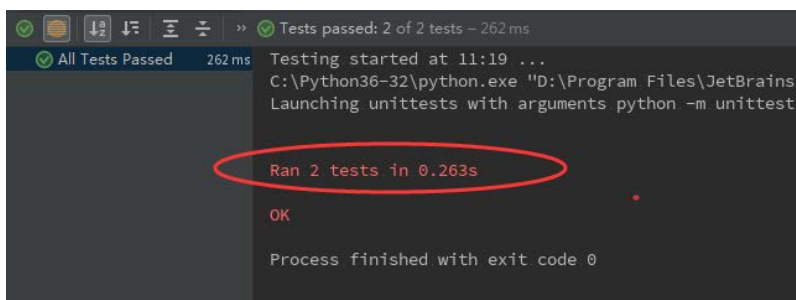


图 13-7-2

13.8 Session 实战

某些使用 Session 处理的系统，登录成功后并没有返回 Session，但是登录成功后的接口请求中包含有这个登录成功后的标识，实例如，图 13-8-1 所示。

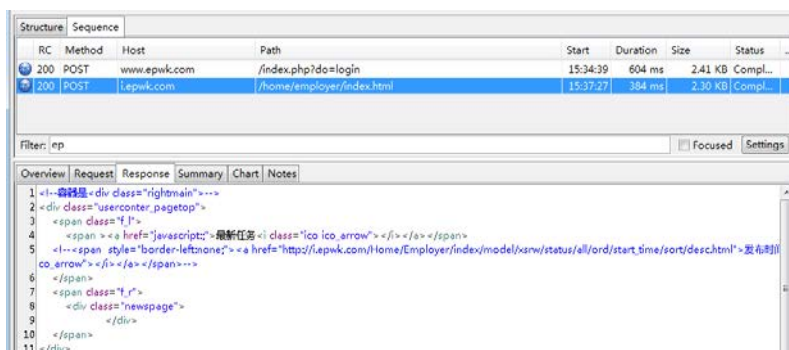


图 13-8-1

在图 13-8-1 中看到发送的请求中，请求头需要带 Cookie，对应的 value 值是登录成功后返回的 SessionID，它的步骤为：

- (1) 登录成功；
- (2) 登录成功后，拿到 SessionID；
- (3) 接口发送请求时，需要带上拿到的 SessionID。

在第三步骤中，如果不带上 SessionID，就会跳转到登录的页面。在这里请求访问 <http://i.epwk.com/home/employer/index.html>，见实现的代码：

```
#!/usr/bin/env python
#coding:utf-8
```

```
#Author:WuYa

import requests
import json
import urllib3
urllib3.disable_warnings()

def getHeaders():
    headers={
        'Content-Type':'application/x-www-form-urlencoded',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81
Safari/537.36'}
    return headers

def employer():
    r=requests.post(
        url='http://i.epwk.com/home/employer/index.html',
        data={'model':'xsrw','status':'all'},
        headers=getHeaders())
    print(r.text)

if __name__ == '__main__':
    employer()
```

在如上代码中，发送 POST 请求查看个人主页发布的任务。由于 cookies 请求参数未带上登录成功后的 SessionID 信息，发送请求后会跳转到登录的页面，运行代码后，如图 13-8-2 所示。

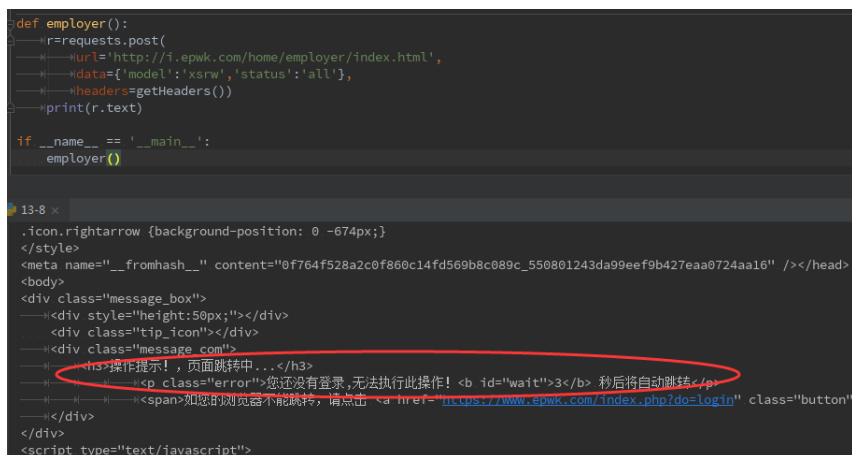


图 13-8-2

在登录成功后返回 Cookies 的信息，在发送发布任务的接口中带上 Cookies 的信息，就能够看到发布任务的信息，而不会跳转到登录的页面。实现的代码为：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import json
import urllib3
urllib3.disable_warnings()

def getHeaders():
    headers={
        'Content-Type':'application/x-www-form-urlencoded',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36'
    }
    return headers

def getData():
    dict1={'formhash':'b4ba6e','txt_account':'13484545195','pwd_password':'60e52c87966078c0b3fab7debb0fc892','login_type':3,'ckb_cookie':0,
    'hdn_refer':'http://www.epwk.com/','txt_code':'','pre':'login','inajax':1}
    return dict1

def login():
    r=requests.post(
        url='https://www.epwk.com/index.php?do=login',
        data=getData(),
        headers=getHeaders(),
        verify=False)
    return r.cookies

def employer():
    r=requests.post(
        url='http://i.epwk.com/home/employer/index.html',
        data={'model':'xsrw','status':'all'},
        headers=getHeaders(),cookies=login())
    print(r.text)
```

```
if __name__ == '__main__':
    employer()
```

注解：在以上代码中，执行登录接口成功后，返回 `r.cookies` 的信息，在发布任务的接口中，在请求的参数中，`cookies` 会带上登录成功后 `login` 接口用例返回的 `SessionID` 信息，也就是用户登录成功后的标识信息 `SessionID`。在 `login` 函数中打印出 `r.cookies` 的信息，内容如下：

```
<RequestsCookieJar[<Cookie
PHPSESSID=1fdb520fb68cb0088ec5c18da5b92eb5fc774fd5    for    .epwk.com/>,
<Cookieaccess_token=08dciXjZh1Rtsc3hHcj4wUViLfVGvcI%2FR%2BPKMKwgOtcTsg8d
3F18yX%2BnYlIDQfk%2Bqujxn0gs2XOOHLclw    for    .epwk.com/>, <Cookie
im_calling_data=e659C3UvRS8icfnxx4crT1huHLdhCKHwQ%2ByvauwpSVKMb61MocXMrd
lXrOq54xsyXIIm5WoLcbv5MQ3fKUqHX1OTtCYMkHIWbPOsmfMjdiHhNb254bI77Euwba32ma
o7%2B6Prr6cHRKXK06Qz3I6HgkU%2F6J73W4%2BpHmWOIrxs5vfowB7pZoViuLEjQJ6QehPz
%2BAqvJV%2Fb7FagRs6cmqCGmiHeqGsJLHSduL%2Bni0ayoNKKx%2BIIeeU0ARPQ28a2EvA9
7IF0r9Y188MXTaicgIWYjr3IAboAvC7DWpAL79O56ggap68cXKf8w47NH7rv8Do7MUISVzA8
rLnCqB4WZhhayS8cuW0SSmQOUw42yloilB4cSCY    for    .epwk.com/>, <Cookie
username=o5c5547ca20dd3    for    .epwk.com/>]>
```

再次运行代码，查看发布任务接口返回的内容，会显示发布任务的信息，而不会重定向跳转到登录的页面，如图 13-8-3 所示。

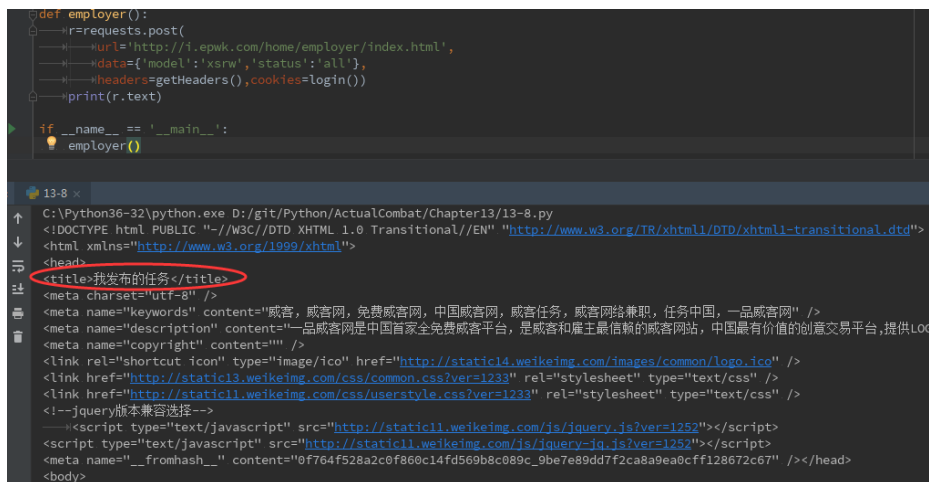


图 13-8-3

下面结合 `unittest` 单元测试框架来完善接口测试用例，代码如下：

```

#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import unittest

def getHeaders():
    headers={
        'Content-Type':'application/x-www-form-urlencoded',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81 Safari/537.36'}
    return headers

def getData():
    dict1={'formhash':'b4ba6e','txt_account':'134****5195','pwd_password':'60e52c87966078c0b3fab7debb0fc892','login_type':3,'ckb_cookie':0,'hdn_refer':'http://www.epwk.com/','txt_code':'','pre':'login','inajax':1}
    return dict1

class SessionTest(unittest.TestCase):
    def login(self):
        r = requests.post(
            url='https://www.epwk.com/index.php?do=login',
            data=getData(),
            headers=getHeaders())
        return r.cookies

    def test_task(self):
        '''我的发布任务'''
        r = requests.post(
            url='http://i.epwk.com/home/employer/index.html',
            data={'model': 'xsrw', 'status': 'all'},
            headers=getHeaders(),
            cookies=self.login())
        self.assertEqual(r.status_code,200)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

执行后界面如图 13-8-4 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-8.py
test_task (__main__.SessionTest)
我的发布任务 ... ok

-----

Ran 1 test in 0.900s

OK

Process finished with exit code 0
|
```

图 13-8-4

13.9 Session 会话对象

Session 会话对象可以在同一个 Session 实例发出的所有请求之间保持 Cookie，同时当你向同一个主机发送多次请求，底层的 TCP 连接将会被重用，从而带来性能上的提升。本节介绍直接通过 Session 会话对象处理多个请求共享 SessionID 的方法，替代 13.8 节所介绍的，把登录成功后获得的 SessionID 存储到文件中的方法，使用 Session 会话对象修改后的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import unittest

def getHeaders():
    headers={
        'Content-Type':'application/x-www-form-urlencoded',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81
Safari/537.36'}
    return headers

def getData():
    dict1={'formhash':'b4ba6e','txt_account':'13484545195','pwd_password':'6
```

```

0e52c87966078c0b3fab7debb0fc892', 'login_type': 3, 'ckb_cookie': 0,

'hdn_refer': 'http://www.epwk.com/', 'txt_code': '', 'pre': 'login', 'inajax':
1}
    return dict1

def login():
    s=requests.Session()
    r=s.post(
        url='https://www.epwk.com/index.php?do=login',
        data=getData(),
        headers=getHeaders())
    return s

def employer():
    r=login().post(
        url='http://i.epwk.com/home/employer/index.html',
        data={'model': 'xsrw', 'status': 'all'},
        headers=getHeaders())
    print(r.text)

if __name__ == '__main__':
    employer()

```

注解：s=requests.Session()是实例化 Session 会话对象，对 Session 类进行实例化对象后，使用实例化返回的对象去发送请求，就会在所有的请求之间保持 cookie，执行 employer 函数后，会先执行登录函数，然后执行 employer 函数。

下面通过 Session 会话对象结合 unittest 单元测试框架，完善接口测试用例，增加对接口执行结果的验证，实现的结果为：

```

#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import unittest

def getHeaders():
    headers={
        'Content-Type': 'application/x-www-form-urlencoded',
        'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81

```



```

Safari/537.36'}
    return headers

def getData():

dict1={'formhash':'b4ba6e','txt_account':'134****5195','pwd_password':'6
0e52c87966078c0b3fab7debb0fc892','login_type':3,'ckb_cookie':0,

'hdn_refer':'http://www.epwk.com/','txt_code':'','pre':'login','inajax':
1}
    return dict1

class SessionTest(unittest.TestCase):
    def s(self):
        '''实例化Session 的对象'''
        return requests.Session()

    def test_login(self):
        '''登录到一品威客'''
        r =self.s().post(
            url='https://www.epwk.com/index.php?do=login',
            data=getData(),
            headers=getHeaders())
        self.assertEqual(r.status_code,200)
        self.assertEqual(r.json()['status'],1)
        self.assertEqual(r.json()['data']['mobile'],'134****5195')

    def test_task(self):
        '''我的发布任务'''
        r =self.s().post(
            url='http://i.epwk.com/home/employer/index.html',
            data={'model': 'xsrw', 'status': 'all'},
            headers=getHeaders())
        self.assertEqual(r.status_code,200)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注释：在如上代码中，在测试类 `SessionTest` 中定义了方法 `s`。对 `requests` 中 `Session` 类实例化并且返回它的对象，这样不管是 `test_login` 接口用例还是 `test_task` 接口用例，可以直接使用 `s` 方法调用对应的请求方法。

执行以上代码后，接口测试用例执行通过，但是显示了警告信息 ResourceWarning: unclosed <socket.socket fd=336, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 8259), raddr=('127.0.0.1', 8888)>, 如图 13-9-1 所示。

```
C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-9.py
test_login (__main__.SessionTest)
登录到一品威客 ... C:\Python36-32\lib\unittest\case.py:601: ResourceWarning: unclosed <socket.socket fd=288,
testMethod()
ok
test_task (__main__.SessionTest)
我的发布任务 ... C:\Python36-32\lib\unittest\case.py:601: ResourceWarning: unclosed <socket.socket fd=336, f
testMethod()
ok

-----
Ran 2 tests in 0.636s

OK

Process finished with exit code 0
```

图 13-9-1

为了解决 ResourceWarning 警告信息的问题，需要导入 warnings，在实例化 Session 类的方法中忽略警告信息，完善后的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import unittest
import warnings

def getHeaders():
    headers={
        'Content-Type':'application/x-www-form-urlencoded',
        'User-Agent':'Mozilla/5.0 (Windows NT 6.1; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.81
Safari/537.36'}
    return headers

def getData():

dict1={'formhash':'b4ba6e','txt_account':'13484545195','pwd_password':'6
0e52c87966078c0b3fab7debb0fc892','login_type':3,'ckb_cookie':0,

'hdn_refer':'http://www.epwk.com/','txt_code':'','pre':'login','inajax':
```

```

1}
    return dict1

class SessionTest(unittest.TestCase):
    def s(self):
        warnings.simplefilter("ignore", ResourceWarning)
        '''实例化 Session 的对象'''
        return requests.Session()

    def test_login(self):
        '''登录到一品威客'''
        r =self.s().post(
            url='https://www.epwk.com/index.php?do=login',
            data=getData(),
            headers=getHeaders())
        self.assertEqual(r.status_code,200)
        self.assertEqual(r.json()['status'],1)
        self.assertEqual(r.json()['data']['mobile'],'13484545195')

    def test_task(self):
        '''我的发布任务'''
        r =self.s().post(
            url='http://i.epwk.com/home/employer/index.html',
            data={'model': 'xsrw', 'status': 'all'},
            headers=getHeaders())
        self.assertEqual(r.status_code,200)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

再次运行代码，可以看到在结果信息中已不显示 `ResourceWarning` 的警告信息，如图 13-9-2 所示。

```

C:\Python36-32\python.exe D:/git/Python/ActualCombat/Chapter13/13-9.py
test_login (__main__.SessionTest)
登录到一品威客 ... ok
test_task (__main__.SessionTest)
我的发布任务 ... ok

-----
Ran 2 tests in 3.203s

OK

Process finished with exit code 0

```

图 13-9-2

13.10 Requests 鉴权实战

Requests 模块提供了对鉴权的处理，结合 10.2.9 节中的内容，客户端发送请求后，服务端响应回复内容 401，响应头中 WWW-Authenticate 对应的内容是 Basic realm="Authentication Required"。在 Requests 库中的请求方法中，提供了 auth 参数来处理鉴权，这需要导入 auth 模块中的 HTTPBasicAuth 类，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests
from requests.auth import HTTPBasicAuth

r=requests.get(
    url='http://localhost:5000/hotel/username/',
    #鉴权的处理
    auth=HTTPBasicAuth('wuya','admin'))

print(r.text)
```

运行以上代码后，服务端返回响应数据如下：

```
{
  "datas": [
    {
      "check in": "2018-03-08 08:20:10",
      "check out": "2018-03-09 14:00:00",
      "identity card": "23012919950425723X",
      "phone": "13484545190",
      "room number": "1104",
      "userid": 1,
      "username": "\u674e\u56db",
      "vpl": "\u4eacAJ3585"
    }
  ]
}
```

在不导入 `HTTPBasicAuth` 类的情况下,对以上代码,也可以直接在请求方法的 `auth` 参数中填写用户的 ID 和密码,修改后的代码如下:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

r=requests.get(
    url='http://localhost:5000/hotel/username/',
    #鉴权的处理
    auth=('wuya','admin'))

print(r.text)
```

13.11 超时处理

在接口测试中,特别是跨产品的接口测试中,例如,你在 A 产品中添加了一条数据,想在 B 产品中查看这个数据是否存在,那么,这个过程属于一个跨产品的过程。这个过程可能需要 1 s,也可能 6 s 或者更长。由于网络等客观因素导致请求超时会报 `requests.exception.Timeout: HTTPConnectionPool` 等异常错误,超时主要处理的是客户端发送请求到服务端的这个过程,超实用到的参数是 `timeout`。例如,请求百度首页,要求 0.01 s 之内连接成功,如果请求超时则报错误信息,实现的代码如下:

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests
```

```
r=requests.get(
    url='http://www.baidu.com/',
    timeout=0.01)

print(r.text)
```

运行以上代码后，连接超时所报的错误信息如图 13-11-1 所示。

```
Traceback (most recent call last):
  File "E:/git/Python/ActualCombat/Chapter12/timeoutTest.py", line 12, in <module>
    timeout=0.01)
  File "C:\Python36-32\lib\site-packages\requests\api.py", line 72, in get
    return request('get', url, params=params, **kwargs)
  File "C:\Python36-32\lib\site-packages\requests\api.py", line 58, in request
    return session.request(method=method, url=url, **kwargs)
  File "C:\Python36-32\lib\site-packages\requests\sessions.py", line 508, in request
    resp = self.send(prep, **send_kwargs)
  File "C:\Python36-32\lib\site-packages\requests\sessions.py", line 618, in send
    r = adapter.send(request, **kwargs)
  File "C:\Python36-32\lib\site-packages\requests\adapters.py", line 496, in send
    raise ConnectTimeout(e, request=request)
requests.exceptions.ConnectTimeout: HTTPConnectionPool(host='www.baidu.com', port=80): Max retries exceeded with url: / (Caused by ConnectTimeoutError(Garlik3 connection.HTTPConnection
```

图 13-11-1

把以上代码中的 `timeout` 参数从 0.01 s 修改为 6 s，也就是说在 6 s 范围内请求连接成功都不会报超时的错误，只有当请求超过 6 s 服务端依然无响应，客户端才会报超时的错误，修改后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

r=requests.get(
    url='http://www.baidu.com/',
    timeout=6)

print(r.text)
```

执行成功后，返回响应内容。

13.12 文件下载

在接口自动化测试中，文件下载是一个比较常见的场景。在 Requests 的库中，默认的情况下，发送下载的请求后，可以通过 stream 参数覆盖这个行为，推迟下载响应直到访问 Response.content 属性，也就是 r=requests.get(request_url, stream=True)。stream 的参数一般都是 True，下载文件的过程中发送下载的请求，把下载的文件内容写入到指定的文件中，导出的数据如图 13-12-1 所示。

名称	车牌号	面额	发券时间	状态	进场时间
接口测试	 陕AJ0002	 1时	02-03 08:23	未使用	-
接口测试	 陕AJ0001	 1时	02-03 08:23	未使用	-

图 13-12-1

实例中下载文件抓取的信息如图 13-12-2 所示。

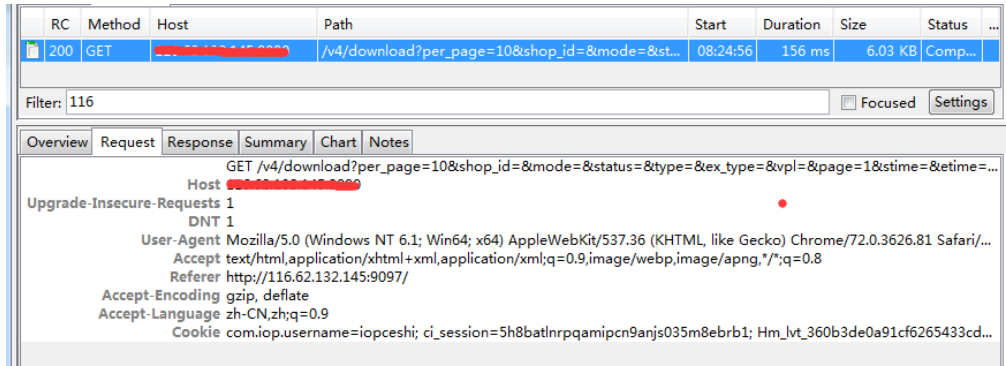


图 13-12-2

在图 13-12-2 中，请求地址是 `http://118.***.***.145:9999/v4/download?per_page=10&shop_id=&mode=&status=&type=&ex_type=&vpl=&page=1&stime=&etime=&token=maXeKfhU5qMvQSF4HWt1549153411144&down=issue`，用户登录成功后，返回 token，在下载的时候对 Token 进行赋值，下载请求发送后，服务端的响应内容如图 13-12-3 所示。

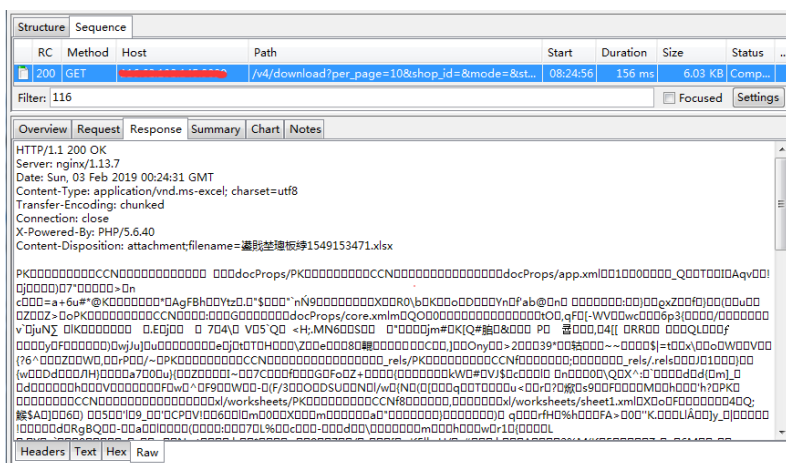


图 13-12-3

在图 13-12-3 中，可以看到服务端返回的响应都是二进制的数，并且 Content-Type 对应的 value 值是 application/vnd.ms-excel; charset=utf8，那么实现的步骤是：

- (1) 客户端发送下载的请求，请求参数中 stream 为 True；
- (2) 服务端收到请求后把响应数据返回给客户端；
- (3) 服务端返回的响应数据是二进制的，经过处理后，把数据写入指定路径的 excel 中，实现的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests

def getHeaders():
    headers={
        'Content-Type':'application/json;charset=UTF-8',
        'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'
    }
    return headers

def login():
    dict1={"username":"66**66","source":"common",
"password":"8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc
```



```

6c92"}
r=requests.post(
    url='http://118.***.***.145:9999/v5/login',
    json=dict1,
    headers=getHeaders())
return r.json()['data']['token']

def download(filepath):
    r=requests.get(

url='http://118.***.***.145:9999/v4/download?per_page=10&shop_id=&mode=&
status=&type=&'

'ex_type=&vpl=&page=1&stime=&etime=&token={0}&down=issue'.format(login()
),
    headers={'Content-Type': 'application/vnd.ms-excel; charset=utf8'})
#判断是否请求成功,如果请求成功,把数据写入到指定的 excel 文件中
if r.status_code==200:
    #二进制的方式把数据写入到文件中
    with open(filepath,'wb') as f:
        for chunk in r.iter_content(chunk_size=1024):
            f.write(chunk)
    return '下载文件成功'

if __name__ == '__main__':
    download('c:/wuya.xlsx')

```

注解：在发送请求时，stream=True 为流式请求，发送请求后进行判断，如果返回的状态码是 200，就把流式请求的内容写入文件中。

运行以上代码后，在 C 盘的目录下生成 download.xlsx，如图 13-12-4 所示。

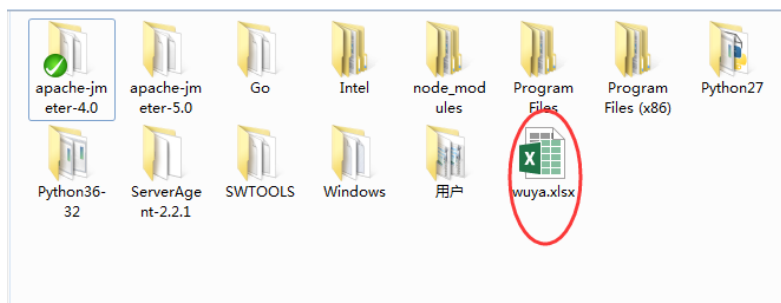


图 13-12-4

在图 13-12-4 中可以看到 wuya.xlsx 已下载，文件的内容如图 13-12-5 所示。

A	B	C	D	E	F	G	H	I
优惠券类型	发放总额	发放记录数						
时长优惠券	2 (小时)	有2条发放记录数						
金额优惠券	0 (元)	有0条发放记录数						
次数优惠券	0 (次)	有0条发放记录数						
时间段优惠券		有0条发放记录数						
折扣优惠券		有0条发放记录数						
下发状态	发券人	车牌	发券方式	发券时间	类别	类型	面额	
未使用	接口测试	陕AJ0002	车牌号发券	2019-02-03 01	预定义	时长	1 小时	
未使用	接口测试	陕AJ0001	车牌号发券	2019-02-03 01	预定义	时长	1 小时	

图 13-12-5

注解：在图 13-12-5 中，可以看到文件内容与页面的截图数据是一致的。

另外一种方式是把下载的文件进行流式处理后，把二进制的内容写入文件中，并且 copy 到指定的目录下。这里会用到 shutil 模块中的 copyfileobj 方法，实现的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests
import shutil

def getHeaders():
    headers={
        'Content-Type':'application/json;charset=UTF-8',
        'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'
    }
    return headers

def login():
    dict1={"username":"66**66","source":"common",
"password":"8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92"}
    r=requests.post(
        url='http://118.***.***.145:9999/v5/login',
        json=dict1,
```

```

        headers=getHeaders())
    return r.json()['data']['token']

def download(filepath):
    r=requests.get(

url='http://118.***.***.145:9999/v5/download?per_page=10&shop_id=&mode=&
status=&type=&'

'ex_type=&vpl=&page=1&stime=&etime=&token={0}&down=issue'.format(login()
),
        headers={'Content-Type': 'application/vnd.ms-excel; charset=utf8'})
    #判断是否请求成功,如果请求成功,把数据写入到指定的 excel 文件中
    if r.status_code==200:
        #二进制的方式把数据写入到文件中
        with open(filepath,'wb') as f:
            r.raw.decode_content = True
            shutil.copyfileobj(r.raw,f)
        return '下载文件成功'

if __name__ == '__main__':
    download('c:/wuya.xlsx')

```

注释：在 Requests 中，默认情况下进行网络请求后，响应内容会立即被下载下来。通过把请求中关键字 `stream` 设置为 `True`，让 Requests 无法将连接释放回连接池，可以延迟响应内容的下载。请求得到响应之后，先判断服务器端返回的状态码是否为 200。如果为 200，就把文件内容以二进制的方式写入到对应的 excel 文件中。

13.13 文件上传

在接口自动化测试中，文件上传的场景也是比较常见的，文件上传中 `Content-Type` 是 `multipart/form-data`，下面以人人网的产品为案例说明这部分的应用。

在人人网登录成功后，上传主页上的图片，上传图片抓取到的信息如图 13-13-1 所示。

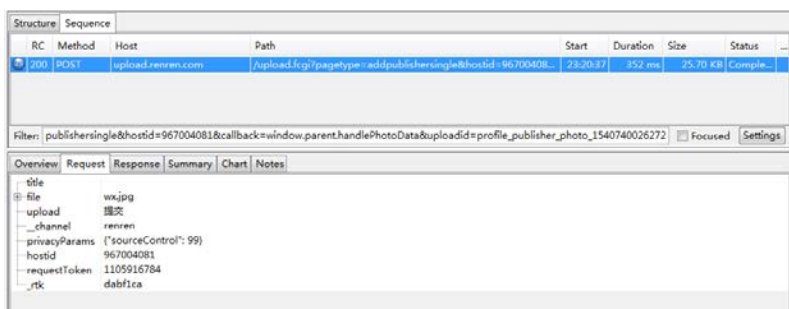


图 13-13-1

在 Charles 中点击 Headers，上传图片的请求头中 Content-Type 对应的 value 值，如图 13-13-2 所示。

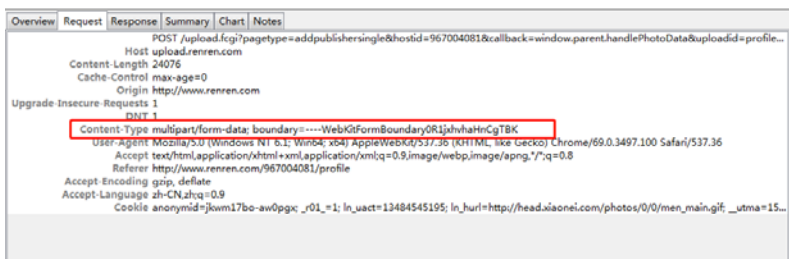


图 13-13-2

在图 13-13-2 中可以看到，图片上传请求参数中存在 file，它对应的是需要上传的图片。在 Requests 中上传文件，可在请求中使用 files 参数并且指明请求的参数名称、上传文件的类型、以及上传文件的路径。关于上传文件的类型，点击 Raw 即可看到 Content-Type 显示出的具体要上传文件的类型，如图 13-13-3 所示。



图 13-13-3

从以上信息可以看出，图片上传中，请求参数中 `files` 对应的值为：`{"file": ("wx.jpg", open("c:/wx.jpg", "rb"), "image/jpeg", {})}`。

在人人网中，要在个人主页上传图片首先要进行登录，登录成功后返回 `Cookies` 的信息。在上传图片的请求中需要带上登录成功后返回的 `Cookies` 信息，上传图片实现的代码如下：

```
#!/usr/bin/env python
#coding:utf-8

#Author:WuYa

import requests

def loginData():
    ''' 登录请求参数 '''
    data = {
        'email': '134****5195',
        'icode': '',
        'origURL': 'http://www.renren.com/home',
        'domain': 'renren.com',
        'key_id': 1,
        'captcha_type': 'web_login',
        'password':
'8d9a71152919613bbe3df9bfa0e1b390eb2b13dd1bdde270c2816cf04dd1b7c5',
        'rkey': 'b4cdc6acc1d36171e3de73dd4676208e',
        'f': 'http%3A%2F%2Fname.renren.com%2F'
    }
    return data

def login():
    r = requests.post(
url='http://www.renren.com/ajaxLogin/login?1=1&uniqueTimestamp=201894216799',
        data=loginData(),
        headers={'Content-Type': 'application/x-www-form-urlencoded'})
    return r.cookies

def uploadData():
    ''' 上传文件请求参数 '''
    data = {
        "upload": "提交",
        "__channel": "renren",
        "privacyParams": '{"sourceControl": 99}',
        'hostid': '967004081',
```

```

        'requestToken': '-1124080368',
        '_rtk': '88c0e36a'}
    return data

def upload():
    '''上传文件方法'''
    r = requests.post(

url='http://upload.renren.com/upload.fcgi?pagetype=addpublishersingle&ho
stid=967004081&'

'callback=window.parent.handlePhotoData&uploadid=profile_publisher_photo
_1540215890321',
    data=uploadData(),
    headers={'Content-Type': 'multipart/form-data'},
    files={"file": ("wx.jpg", open("c:/wx.jpg", "rb"), "image/jpeg",
{}})},
    cookies=login())
    print(r.status_code)
    print(r.text)

if __name__ == '__main__':
    upload()

```

13.14 Requests 接口测试实战

前面详细地介绍了 Requests 在接口测试中的应用，下面结合具体的实例介绍 Requests 在接口自动化测试中的实战应用。首先在 PyCharm 中创建项目 api，在该项目中创建 data 文件夹，用来存放测试数据。report 文件存放测试报告，page 包用来编写对象层的代码，testCase 包是存储各个模块的测试代码，allTests.py 模块是执行所有测试模块的文件，如图 13-13-1 所示。

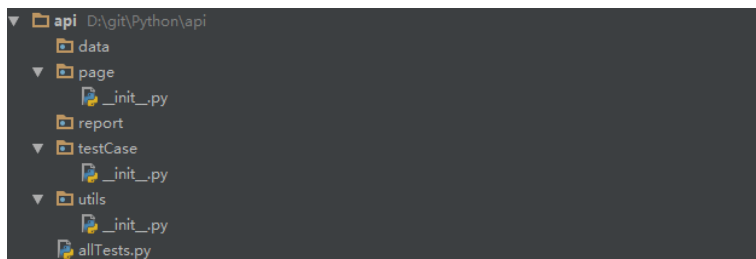


图 13-14-1

实例任务为登录系统成功后，创建用户、查询用户、冻结用户再激活用户，最后删除用户。首先在 `page` 包中创建 `user.py` 模块文件，在 `user.py` 文件中编写对象层的代码，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import requests

def getHeaders():
    '''返回请求头'''
    headers= {'Content-Type':'application/json; charset=UTF-8',
              'Parkingwang-Client-Source':'ParkingWangAPIClientWeb'}
    return headers

def post(url,data):
    '''
    对 post 请求方法进行二次封装
    : param url:请求地址
    : param data:请求参数
    '''
    r=requests.post(
        url=url,
        json=data,
        headers=getHeaders(),
        timeout=6)
    return r
```

注解：在 `user.py` 模块中，`getHeaders` 函数返回请求头，`post` 函数是对 `Requests` 库中的 `post` 请求方法进行了二次封装。

在 `testCase` 包中创建 `test_user.py` 模块文件，在该文件中编写登录成功后，获取 `Token` 并且存储到文件中，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import requests
```

```

import time as t
import os

from page.user import *

class TestUserApi(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        t.sleep(1)

    @classmethod
    def tearDownClass(cls):
        pass

    def test_user_api_001(self):
        ''' 登录业务: 登录系统 '''
        r=post(
            'http://118.***.***.145:9999/v5/login',
            {"username": "6666666666",
             "password": "8144ed050cd8d053f24a1e179d7529e17c3a2ba9cfcfcd7d3bda9ec6a8156758"})
        self.assertEqual(r.status_code, 200)
        self.assertEqual(r.json()['status'], 0)
        self.assertEqual(r.json()['data']['name'], '6666666666')
        with open(os.path.join(os.path.dirname(
            os.path.dirname(__file__)), 'data', 'token'), 'w') as f:
            f.write(r.json()['data']['token'])

    @property
    def getToken(self):
        ''' 获取登录成功后的 token '''
        with open(os.path.join(os.path.dirname(
            os.path.dirname(__file__)), 'data', 'token'), 'r') as f:
            return f.read()

    def test_user_api_002(self):
        ''' 登录业务: 查看登录成功后的用户信息 '''
        r=post(
            'http://118.***.***.145:9999/v5/infoGet',
            {'token': self.getToken})
        self.assertEqual(r.status_code, 200)
        self.assertEqual(r.json()['status'], 0)
        self.assertEqual(r.json()['data']['username'], '6666666666')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```


注解：在测试模块 `test_user.py` 中，测试类 `TestUserApi` 继承了 `unittest` 模块中的 `TestCase` 类，编写了登录成功后把返回的 `Token` 写入 `data` 文件夹下 `token` 文件中，方法 `getToken` 读取 `data` 文件下 `token` 文件的内容。第二个测试用例是在请求中直接读取文件中 `token` 的内容，并作为参数上传。以上测试代码，不管是向文件中写入内容、读取文件中的内容，或是断言，都有可优化的空间。运行以上代码后的结果如图 13-14-2 所示。

```
C:\Python36-32\python.exe D:/git/Python/api/testCase/test_user.py
test_user_api_001 (__main__.TestUserApi)
登录业务-登录系统 ... ok
test_user_api_002 (__main__.TestUserApi)
登录业务-查看登录成功后的用户信息 ... ok

-----

Ran 2 tests in 1.408s

OK
```

图 13-14-2

下面对以上代码进行优化。把文件的路径写在 `utils` 包中的 `helper` 方法中，将测试用例中对 `HTTP` 协议的断言和业务状态码的断言，单独提取出来写一个方法直接调用。`utils` 包中 `htlper.py` 文件的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import os

class Helper(object):
    '''公共方法'''

    def base_dir(self,filePath,folder='data'):
        '''
        返回公共路径
        : param folder:文件夹
        : param filePath:文件名称
        '''
        return os.path.join(os.path.dirname(os.path.dirname(__file__)),
                             folder,filePath)
```

注解：在 `Helper` 类中新增了 `base_dir` 方法，返回公共路径。

优化后的 `test_user.py` 模块的测试代码，主要对断言进行了分离并继承了 `Helper` 类，在 `token` 写入和读取中直接调用 `base_dir` 的方法，`test_user.py` 模块优化后的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import requests
import time as t
import os

from page.user import *
from utils.helper import *

class TestUserApi(unittest.TestCase,Helper):
    @classmethod
    def setUpClass(cls):
        t.sleep(1)

    @classmethod
    def tearDownClass(cls):
        pass

    def statusCode(self,r):
        '''对HTTP 状态码和业务状态码校验'''
        self.assertEqual(r.status_code,200)
        self.assertEqual(r.json()['status'],0)

    def test_user_api_001(self):
        '''登录业务:登录系统'''
        r=post(
            'http://118.***.***.145:9999/v5/login',
            {"username":"66**66",
            "password":"8144ed050cd8d053f24a1e179d7529e1
7c3a2ba9cfcfcd7d3bda9ec6a8156758"})
        self.statusCode(r)
        self.assertEqual(r.json()['data']['name'],'66**66')
        with open(self.base_dir('token'),'w') as f:
```

```
f.write(r.json()['data']['token'])

@property
def getToken(self):
    ''' 获取登录成功后的token'''
    with open(self.base_dir('token'),'r') as f:
        return f.read()

def test_user_api_002(self):
    ''' 登录业务: 查看登录成功后的用户信息'''
    r=post(
        'http://118.***.***.145:9999/v5/infoGet',
        {'token':self.getToken})
    self.statusCode(r)
    self.assertEqual(r.json()['data']['username'],'66**66')

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

在 test_user.py 模块中的测试用例中，请求地址和请求数据并没有被分离出来，如果请求地址和请求数据发生变化，则维护起来成本较高。这时可把这部分分离到 excel 文件中，在 excel 中进行统一的维护。在 data 文件夹下创建 data.xlsx 文件，并且写入内容，data.xlsx 文件中的内容如图 13-14-3 所示。

模块	url	data
登录业务-登录	http://118.***.***.145:9999/v5/login	{"username":"6666666666","source":"common","password":"8d969eef6
登录业务-查看用户信息	http://118.***.***.145:9999/v5/infoGet	{"token":"maASPWNHCsg2KBdM38B1549786274081"}

图 13-14-3

接下来在 helper.py 模块中编写读取 excel 数据的方法，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import os
import xlrd

class Helper(object):
```

```

''' 公共方法'''

def base_dir(self, filePath, folder='data'):
    '''
    返回公共路径
    :param folder: 文件夹
    :param filePath: 文件名称
    '''
    return os.path.join(os.path.dirname(os.path.dirname(__file__)),
                        folder, filePath)

def readExcel(self, rowx, filePath='data.xlsx'):
    '''
    读取 excel 中数据并且返回
    :param filePath: xlsx 文件名称
    :param rowx: 在 excel 中的行数
    '''
    book = xlrd.open_workbook(self.base_dir(filePath))
    sheet = book.sheet_by_index(0)
    return sheet.row_values(rowx)

```

注解：在类 `Helper` 中，首先编写处理文件目录的 `base_dir` 的方法，然后导入 `xlrd` 的库，编写读取 `excel` 中的数据；在方法 `readExcel` 中，首先读取 `excel` 的文件，然后读取 `excel` 文件中第一个 `sheet` 中的数据，并且按行的形式返回。

在以上 `excel` 文件中，虽然请求地址和请求数据每个接口都不一样，但是请求地址不管是哪个接口，都在第 `N` 行第二列，请求数据在第 `N` 行第三列。在 `Helper` 类中新增获取请求方法和获取请求地址的方法，则修改后的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import os
import xlrd
import json

class Helper(object):
    '''公共方法'''

```

```

def base_dir(self, filePath, folder='data'):
    '''
    返回公共路径
    : param folder:文件夹
    : param filePath:文件名称
    '''
    return os.path.join(
        os.path.dirname(
            os.path.dirname(__file__)),
        folder, filePath)

def readExcel(self, rowx, filePath='data.xlsx'):
    '''
    读取 excel 中数据并且返回
    : param filePath:xlsx 文件名称
    : param rowx:在 excel 中的行数
    '''
    book=xlrd.open_workbook(self.base_dir(filePath))
    sheet=book.sheet_by_index(0)
    return sheet.row_values(rowx)

def getUrl(self, rowx):
    '''
    获取请求地址
    : param rowx:在 excel 中的行数
    '''
    return self.readExcel(rowx)[1]

def getData(self, rowx):
    '''
    获取数据并且返回
    : param rowx:在 excel 中的行数
    '''
    return json.loads(self.readExcel(rowx)[2])

```

在 test_user.py 中对测试用例进行修改，请求地址和请求数据已经分离到 excel 文件中，直接调用对应的方法，该测试类的代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest

```

```

import requests
import time as t
import os

from page.user import *
from utils.helper import *

class TestUserApi(unittest.TestCase,Helper):
    @classmethod
    def setUpClass(cls):
        t.sleep(1)

    @classmethod
    def tearDownClass(cls):
        pass

    def statusCode(self,r):
        ''' 对HTTP 状态码和业务状态码校验'''
        self.assertEqual(r.status_code,200)
        self.assertEqual(r.json()['status'],0)

    def test_user_api_001(self):
        ''' 登录业务: 登录系统'''
        r=post(self.getUrl(1),self.getData(1))
        self.statusCode(r)
        self.assertEqual(r.json()['data']['name'],'66**66')
        with open(self.base_dr('token'),'w') as f:
            f.write(r.json()['data']['token'])

    @property
    def getToken(self):
        ''' 获取登录成功后的token'''
        with open(self.base_dr('token'),'r') as f:
            return f.read()

    def test_user_api_002(self):
        ''' 登录业务: 查看登录成功后的用户信息'''
        r=post(self.getUrl(2),self.getData(2))
        self.statusCode(r)
        self.assertEqual(r.json()['data']['username'],'66**66')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在以上代码中可以看到，在两个接口用例中，已经分离出了请求地址和请求数据，直接调用的时候，只需要注意对应的数据在 excel 文件中的行数，指定具体的索引就可以了（索引是从 0 开始，所以第 N 行，索引就是 $N-1$ ）。运行以上代码后结果如图 13-14-4 所示。

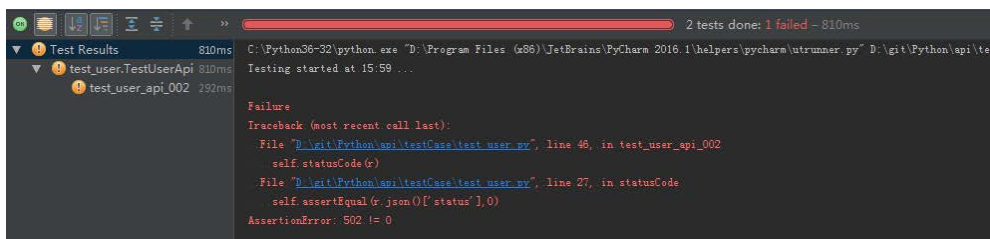


图 13-14-4

在图 13-14-4 中可以看到，第二个接口用例失败，返回的业务状态码为 502，也就是无效的 Token。

在以上测试模块执行中，第二个接口用例之所以返回失败，是因为请求参数中的 Token 与登录成功后的 Token 不一致导致报错。那么也就意味着每次登录成功后，都需要用新的 Token 为 excel 文件中的 Token 进行赋值，让后续发送请求的 Token 与登录成功后的 Token 保持一致，这样才不会出现 502 的错误。操作步骤为：

- (1) 读取 excel 文件中的数据；
- (2) 对请求参数中的 token 进行赋值；
- (3) 返回赋值后的数据。

再次修改 test_user.py 模块的代码，新增 setToken 的方法，在登录成功后的请求参数中直接调用 setToken 的方法，代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import requests
import time as t
import os
import json
```

```

from page.user import *
from utils.helper import *

class TestUserApi(unittest.TestCase,Helper):
    @classmethod
    def setUpClass(cls):
        t.sleep(1)

    @classmethod
    def tearDownClass(cls):
        pass

    def statusCode(self,r):
        '''对HTTP 状态码和业务状态码校验'''
        self.assertEqual(r.status_code,200)
        self.assertEqual(r.json()['status'],0)

    def test_user_api_001(self):
        '''登录业务:登录系统'''
        r=post(self.getUrl(1),self.getData(1))
        self.statusCode(r)
        self.assertEqual(r.json()['data']['name'],'66**66')
        with open(self.base_dir('token'),'w') as f:
            f.write(r.json()['data']['token'])

    @property
    def getToken(self):
        '''获取登录成功后的token'''
        with open(self.base_dir('token'),'r') as f:
            return f.read()

    def setToken(self,rowx):
        '''对excel 中的请求参数token 重新赋值'''
        dict1=self.getData(rowx)
        dict1['token']=self.getToken
        return dict1

    def test_user_api_002(self):
        '''登录业务:查看登录成功后的用户信息'''
        r=post(self.getUrl(2),self.setToken(2))
        self.statusCode(r)
        self.assertEqual(r.json()['data']['username'],'66**66')

if __name__ == '__main__':
    unittest.main(verbosity=2)

```


注解：在以上代码中新增 `setToken` 方法，在第二个接口用例中直接调用 `setToken` 方法，执行成功。

接下来依次编写添加用户、冻结用户、激活用户和删除用户的业务，excel 文件中的数据如图 13-14-5 所示。

模块	url	data
登录业务-登录	http://192.168.1.15:8080/v5/login	{ "username": "6666666666", "source": "common", "password": "123456" }
登录业务查看用户信息	http://192.168.1.15:8080/v5/infoGet	{ "token": "maASPWNHCsg2KBdM38B1549786274081" }
用户管理业务-添加用户	http://192.168.1.15:8080/v5/shopAdd	{ "user_type": 0, "status": 0, "num_limit": 0, "username": "666666", "password": "123456" }
用户管理业务-查询用户	http://192.168.1.15:8080/v5/shopRecords	{ "per_page": 10, "page": 1, "name": "接口测试", "token": "maASPWNHCsg2KBdM38B1549786274081" }
用户管理业务-冻结用户	http://192.168.1.15:8080/v5/shopEdit	{ "id": 751167, "username": "666666", "name": "接口测试", "user_type": 0, "status": 0, "num_limit": 0 }
用户管理业务-激活用户	http://192.168.1.15:8080/v5/shopEdit	{ "id": 751167, "username": "666666", "name": "接口测试", "user_type": 0, "status": 0, "num_limit": 0 }
用户管理业务-删除用户	http://192.168.1.15:8080/v5/shopEdit	{ "id": 751167, "username": "666666", "name": "接口测试", "user_type": 0, "status": 0, "num_limit": 0 }

图 13-14-5

在 `test_user.py` 模块中编写用户管理业务的接口用例，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

import unittest
import requests
import time as t
import os
import json

from page.user import *
from utils.helper import *

class TestUserApi(unittest.TestCase,Helper):
    @classmethod
    def setUpClass(cls):
        t.sleep(1)

    @classmethod
    def tearDownClass(cls):
        pass

    def statusCode(self,r):
        '''对HTTP 状态码和业务状态码校验'''
        self.assertEqual(r.status_code,200)
```

```

self.assertEqual(r.json()['status'],0)

def test_user_api_001(self):
    ''' 登录业务: 登录系统 '''
    r=post(self.getUrl(1),self.getData(1))
    self.statusCode(r)
    self.assertEqual(r.json()['data']['name'],'66**66')
    with open(self.base_dir('token'),'w') as f:
        f.write(r.json()['data']['token'])

@property
def getToken(self):
    ''' 获取登录成功后的token '''
    with open(self.base_dir('token'),'r') as f:
        return f.read()

def setToken(self,rowx):
    '''
    对excel 中的请求参数token 重新赋值
    : param rowx: 在excel 中的行数
    '''
    dict1=self.getData(rowx)
    dict1['token']=self.getToken
    return dict1

def test_user_api_002(self):
    ''' 登录业务: 查看登录成功后的用户信息 '''
    r=post(self.getUrl(2),self.setToken(2))
    self.statusCode(r)
    self.assertEqual(r.json()['data']['username'],'66**66')

def test_user_api_003(self):
    ''' 用户管理业务: 添加用户 '''
    r=post(self.getUrl(3),self.setToken(3))
    self.statusCode(r)

def test_user_api_004(self):
    ''' 用户管理业务: 用户查询 '''
    r=post(self.getUrl(4),self.setToken(4))
    self.statusCode(r)
    self.assertEqual(r.json()['data']['records'][0]['name'],
'666666')

@property
def getUserID(self):

```

```

''' 获取用户的ID'''
r = post(self.getUrl(4), self.setToken(4))
return r.json()['data']['records'][0]['id']

def setTokenUserID(self,rowx):
'''
对excel 中的请求参数token,用户ID 重新赋值
: param rowx: 在excel 中的行数
'''
dict1=self.getData(rowx)
dict1['token']=self.getToken
dict1['id']=self.getUserID
return dict1

def test_user_api_005(self):
''' 用户管理业务:冻结用户'''
r=post(self.getUrl(5),self.setTokenUserID(5))
self.statusCode(r)

def test_user_api_006(self):
''' 用户管理业务:验证用户已冻结'''
r = post(self.getUrl(4), self.setToken(4))
self.statusCode(r)
self.assertEqual(r.json()['data']['records'][0]['status'],1)

def test_user_api_007(self):
''' 用户管理业务:激活用户'''
r = post(self.getUrl(6), self.setTokenUserID(6))
self.statusCode(r)

def test_user_api_008(self):
''' 用户管理业务:验证用户已激活'''
r = post(self.getUrl(4), self.setToken(4))
self.statusCode(r)
self.assertEqual(r.json()['data']['records'][0]['status'], 0)

def test_user_api_009(self):
''' 用户管理业务:删除用户'''
r = post(self.getUrl(7), self.setTokenUserID(7))
self.statusCode(r)

def test_user_api_010(self):
''' 用户管理业务:验证用户已删除'''
r = post(self.getUrl(4), self.setToken(4))
self.statusCode(r)

```

```

        self.assertEqual(r.json()['data']['tn'], 0)

if __name__ == '__main__':
    unittest.main(verbosity=2)

```

注解：在冻结用户、激活用户和编辑用户中，需要使用到用户的 ID，但是用户的 ID 是一个动态参数，所以需要在用户添加成功后，在用户列表中获取用户的 ID，用到的方法是 `getUserID`。不管是冻结用户、激活用户还是编辑用户，读取 excel 文件中的数据后，都需要先对动态参数 `token` 和 `id` 参数进行赋值，这样就可以保证在冻结用户、激活用户和编辑用户中，动态参数 `token` 和 `id` 参数是最新的，这里用到的方法是 `setTokenUserID()`。在接下来的冻结用户、激活用户和编辑用户中，直接调用该方法并指明具体的行就可以了。在以上代码中，不管是冻结用户还是激活用户，客户端发送请求后，服务器返回的响应数据中并没有返回用户的状态，也就是说冻结用户和激活用户的接口执行成功后，无法保障用户的状态是否真的冻结或者激活。为了解决该问题，可在冻结（激活）用户后，在用户列表中查询用户的状态，根据用户的状态变化来验证冻结操作或者激活操作是否生效。

在以上代码中，还可以把每个测试用例添加到测试套件中，通过执行每个测试套件来达到批量执行所有测试用例的目的，在项目 `api` 中新创建模块 `allTests.py` 文件，项目目录如图 13-14-6 所示。



图 13-14-6

在模块 `allTetss.py` 文件中编写获取所有测试用例的方法，代码如下：

```

#!/usr/bin/env python
#-*-coding:utf-8-*-

#author:wuya

```

```
import unittest

def allTestCases():
    ''' 获取 testCase 包中所有的测试模块'''
    suite=unittest.defaultTestLoader.discover(
        start_dir='D:/git/Python/api/testCase',
        pattern='test_*.py',
        top_level_dir=None
    )
    return suite
```

注解：使用 `discover` 方法可以获取所有的测试用例。第一个参数 `start_dir` 是指执行哪个包中的测试模块，这里所有的测试模块都存放在 `testCase` 包中，所以 `start_dir` 的参数为 `testCase` 的包目录；第二个参数 `pattern` 是指定 `testCase` 包中测试用例文件的匹配规则，虽然这个实例中，`testCase` 包中只有模块 `test_user.py` 文件，但是，以后在 `testCase` 包中可能还有其他模块的测试模块文件，这里使用了正则表达式 `test_*.py`，这样可以获取 `testCase` 包中所有以 `test_` 开头的测试模块文件。执行 `allTestCases` 函数，输出结果如下：

```
<unittest.suite.TestSuite tests=[<unittest.suite.TestSuite
tests=[<unittest.suite.TestSuite tests=[<test_user.TestUserApi
testMethod=test_user_api_001>, <test_user.TestUserApi
testMethod=test_user_api_002>, <test_user.TestUserApi
testMethod=test_user_api_003>, <test_user.TestUserApi
testMethod=test_user_api_004>, <test_user.TestUserApi
testMethod=test_user_api_005>, <test_user.TestUserApi
testMethod=test_user_api_006>, <test_user.TestUserApi
testMethod=test_user_api_007>, <test_user.TestUserApi
testMethod=test_user_api_008>, <test_user.TestUserApi
testMethod=test_user_api_009>, <test_user.TestUserApi
testMethod=test_user_api_010>]>]>]>
```

注解：可以看到输出的内容为 `test_user.py` 模块中的所有的测试用例。

目录最好使用 `os` 模块来处理。继续完善 `allTests.py` 模块文件，新增测试报告的生成逻辑，实现的代码如下：

```
#!/usr/bin/env python
#-*-coding:utf-8-*-
```

```

#author:wuya

import unittest
import os
import HTMLTestRunner
import time

def allTestCases():
    ''' 获取 testCase 包中所有的测试模块'''
    suite=unittest.defaultTestLoader.discover(
        start_dir=os.path.join(os.path.dirname(__file__),'testCase'),
        pattern='test_*.py',
        top_level_dir=None
    )
    return suite

def getNowTime():
    ''' 获取当前时间'''
    return time.strftime('%Y_%m_%d %H_%M_%S')

def run():
    fp=open(os.path.join(os.path.dirname(__file__),
'report',getNowTime()+'.report.html'),'wb')
    HTMLTestRunner.HTMLTestRunner(
        stream=fp,
        title='接口自动化测试报告',
        description='基于 Python 语言的接口自动化测试实战',
    ).run(allTestCases())

if __name__ == '__main__':
    run()

```

注解：在以上实例中，在函数 `allTestCases` 中返回了 `testCase` 包中所有以 `test` 开头的模块文件，函数 `getNowTime` 用来获取当前时间，在函数 `run` 中指定测试报告生成后存放的目录文件及测试报告的名称。最后在主函数中执行 `run` 函数。

编写完成后，在 Jenkins 平台中创建 API 的 Job，并且执行该自动化测试的所有接口测试用例。在 Job 的配置中，构建后选择 Allure Report 的测试报告形式，创建后的 Job 如图 13-14-7 所示。



图 13-14-7

点击立即构建，执行 API 接口自动化测试用例，执行成功后在 Job 的视图中点击 Allure Report，显示执行的结果报告，如图 13-14-8 所示。



图 13-14-8

在报告中可以看到执行用例的总数、成功率和失败率。

至此，一个完整的接口自动化测试实战介绍完成。后期不管是版本的迭代还是更新，在测试环境合并代码后，可以直接执行该接口测试用例以验证合并代码对功能的影响。这样就不需要每次合并代码后，用手工的方式去验证系统的功能，而且接口测试用例执行的时间很短，只需几分钟的时间就可以看到执行的结果。即使用于生产环境，某个版本上线后，我们只需要在 excel 中把测试环境地址修改为线上环境地址，然后直接执行测试，这样就可以让自动化测试来承担冒烟测试的任务，进而实现一套代码在测试环境和生产环境的双执行。通过这样的方式，提高了测试的执行效率，从真正意义上把测试人员解放出来。

主要参考文献

- [1] （日）上野宣著. 图解 HTTP. 于均良译. 北京：人民邮电出版社，2014.
- [2] （印）Unmesh Gundecha 著. Learning Selenium Testing Tools with Python. Birmingham: Packt Publishing, 2014.
- [3] 周伟，宗杰等编著. Python 开发技术详解. 北京：机械工业出版社，2009.
- [4] （美）Manoj Hans 著. Appium Essentials (English Edition). Birmingham: Packt Publishing, 2015.
- [5] （美）vid Gourley, Brian Totty 著. HTTP 权威指南. 陈涓，赵振平译. 北京：人民邮电出版社，2012.
- [6] （美）Rakesh Vidya Chandra, Bala Subrahmanyam Varanasi. Python Requests Essentials. Birmingham: Packt Publishing, 2015.